Techniques for Assistance in Streamline and Stream Surface Visualizations

Thesis

Presented in Partial Fulfillment of the Requirements for Honors Research Distinction in the College of Engineering at The Ohio State University

By

Ross Vasko

Department of Computer Science and Engineering

The Ohio State University

2017

Thesis Committee:

Rephael Wenger, Advisor

Han-Wei Shen

Abstract

Flow fields provide a mathematical model of fluid flows and arise in various fields of science and engineering. Fluid flows are often visualized with streamlines and stream surfaces. A streamline is a curve that is tangent everywhere to the flow field and a stream surface is a surface that is tangent everywhere to the flow field. These geometric techniques can provide the user with a clear, intuitive visualization if applied effectively. Visualizations with either technique are not trivial to create. Streamline visualizations often suffer from sampling or occlusion issues while the construction of stream surfaces themselves is difficult and time consuming. Here we provide methods to automatically place streamlines in the flow field and to automatically create a stream surface seeding curve.

Streamline visualizations should not be created with arbitrary seedings. An arbitrary seeding can very easily miss features in the flow field or create an unhelpful visualization due to clutter. We describe a method of measuring the complexity of streamlines based on fractal dimension in order to assist the user in creating a seeding of streamlines. A scalar grid is also made from the complexity measurements and can be visualized to further assist the user in locating interesting regions of the flow field.

Stream surfaces are difficult to generate because of the challenge of creating a "good" seeding curve for the surface. A streamline is integrated from a single point, while a curve to integrate is required to create a stream surface. This curve is not

inherently determined by the flow field and must be created by the user. We describe desirable properties to have in a seeding curve and then provide an algorithm that generates seeding curves based on these properties. We also provide a technique based on similarity measurements to filter stream surfaces from the visualization to reduce clutter.

To my sister, Jord.

Acknowledgments

The individual who deserves my most sincere gratitude is my advisor Dr. Rephael Wenger. Thank you for your endless patience as you spent countless hours guiding me through my undergraduate studies. You were always there to assist me with obstacles in my work as well as obstacles in my life. My experience here would not nearly have been as positive without your influence. You made my four years at Ohio State the best they could be. You brought out the best in me.

I also must thank the other member of my thesis committee, Dr. Han-Wei Shen. Thank you for all of the knowledge on flow visualization you provided me with and for everyone in your lab that you introduced me to. I benefited an incredible amount as a researcher from your insight during every one of our meetings.

Thank you to my instructors Dr. Tamal Dey, Dr. Hiranmoy Easwaran, Dr. Barry Minemyer, and Dr. Anastasios Sidiropoulos for helping me further develop my interests in computer science. The courses that I took with all of you were among the best. A combination of each of your lectures and the conversations that I had with all of you helped me find my interest in theory and algorithms. Being around all of you inspired me to be better.

I will never forget those who first influenced me and helped me find my passion for computer science. I sincerely thank Mr. Michael Barile, Mr. Andrew Bruening, Mrs. Heather Klein, Mr. Kevin Klein, Mrs. Laura Lamberty, and Mr. Michael Scott

for providing an environment for me in which I could find something that I love when I felt the most confused. Thank you all for going out of your way to entertain my curiosity and for showing me how fun learning can be, especially Mr. Michael Barile.

Mr. Barile, I look back at the all time we spent together with incredible fondness. You were a role model of mine early on due to your passion for learning and teaching. You remain one of the first people that comes to my mind when I want to tell someone good news. I learned so much from every interaction I had with you and I look forward to continuing to learn from you.

I also thank my family for all of their support in each of their own unique ways. I thank my parents, Jo and Thad, for their encouragement at each stage of my life. You each did your best to always look out for me and I am forever grateful for everything that you both did for me. I thank my sister, Jord, for always being my best friend. You understand me in ways that I do not think that anyone else does. I hold you and your thoughts at the highest level of respect. I am also grateful for David Sullivan being there early in my life to help develop my curiosity. You would love to see the puzzles I work on now.

Lastly, I need to thank anyone who I have come in contact with through Baker, Dreese, Frambes, or River for always being there to remind me to have fun, slow down, and laugh. Thank you for all of the different, unique perspectives that you provided me with that I would not have been able to gain from a text or a lecture. Just as my educators showed me how important learning is to me, you have all taught me how important people and experiences are to me. Each of you are brilliant in ways that I will never be. You all deserve the best.

Table of Contents

List of Tables

Table **Page**

List of Figures

Chapter 1: Introduction

Fluid dynamics is an important part of understanding weather patterns, how air moves over vehicles, and how blood moves through the body. A flow field is a mathematical model of these fluid flows. The visualization of flow fields is crucial in order to allow users to understand their fluid data. It is popular to visualize flow fields with geometric techniques, specifically by integrating through the flow field to create streamlines or stream surfaces. A streamline is the path of a single point integrated through flow field. A stream surface is the surface that is created by integrating some curve through the flow field. Both streamlines and stream surfaces are tangent everywhere to the flow field. However, streamlines and stream surfaces each have their own problems when applied to visualize a data set. We provide two complementary techniques, one for streamlines and one for stream surfaces, to aid in the use of streamlines and stream surfaces in flow visualization.

Our first technique guides the user in a streamline visualization by highlighting regions of interest in the flow and by filtering streamlines. Advances in simulations allow users to create very detailed flow simulations, which become a considerable size especially while in 3D. The large size of these data sets make manual exploration unreasonable. It is far too consuming for a user to manually place streamlines in large data set. It cannot be ensured that the user has captured each region of interest

with the streamlines without examining the data set. The user cannot address this issue with a dense seeding of streamlines in the data set because a dense seeding will create a cluttered visualization. Therefore, an intelligent automatic placement of streamlines is necessary.

In our approach, we first create a dense seeding of streamlines and then measure their complexity using the local box counting ratio. The local box counting ratio is an approximation of the box counting dimension, which quantifies how densely a set fills a space. The dense seeding of streamlines is then filtered by complexity values to allow the user to only see regions of flow that exhibit certain space filling properties. Intuitively, streamlines that fill a space more densely correspond to turbulent flow or vortices, while streamlines that do not fill a space densely correspond to more laminar flow. Additionally, we create a scalar grid over the flow field which quantifies the complexity of the flow at each point in the flow field. Regions with higher scalar values correspond to regions of flow with higher complexity. Applying visualization techniques to this scalar grid representing flow complexity also helps guide the user in finding interesting or complex regions.

Our second technique generates and filters stream surfaces near a point chosen by the user. A significant difficulty in the use of stream surfaces is the choice of curve to be integrated through the flow field. The curve to be integrated is independent of the flow, so a curve must be chosen by the user. However, choosing an arbitrary seeding curve can lead to unhelpful or degenerate stream surfaces. It is also unreasonable for a user to manually construct a seeding curve.

Our algorithm to choose stream surface seeding curves attempts to choose a curve that extracts features from the flow field. We generate a curve to integrate through

the flow to create a stream surface by taking a point as input from the user and then integrating a curve from that point. The curve is integrated in such a manner so that the change in vectors in the flow along the seeding curve is minimized, based on the Jacobian matrix. Intuitively, minimizing the change in vectors along the seeding curve is meant to capture a single feature of the flow field. Additionally, we restrict the generation of the initial seeding curve to a plane that is orthogonal to the flow. This helps avoid creating degenerate stream surfaces.

Our stream surface seeding curve algorithm can be used to generate multiple seeding curves around the initial point provided by the user. The resulting stream surfaces are the filtered to prevent clutter using similarity measurements using the Fréchet distance. The Fréchet distance is a measurement of similarity between curves that considers the order of points in the curve. We measure the Fréchet distance between seeding curves and streamlines from the starting location of each seeding curve to approximate which stream surfaces will be similar. We then remove the similar stream surfaces to prevent the visualization from being cluttered while retaining the distinct features in the visualization.

The contributions of this thesis are the following:

1. Application of fractal dimensions to measuring the complexity of streamlines;
2. Method to filter streamlines by fractal dimension complexity measurements;
3. Method of generating stream surface seeding curves based on vector similarity;
4. Application of similarity measurements to filter stream surfaces.

Chapter 2: Background

2.1 Flow fields

As previously introduced, a flow field is a mathematical model of a fluid flow. Formally, a *flow field* is a continuous function $V : D \to \mathbb{R}^n$, where $D \subseteq \mathbb{R}^n$. If $p \in D$, let $V(p)$ denote the vector at point p. The vector at each point in the flow describes the rate and direction of mass transport of the flow. Flow fields are either time independent (steady) or time dependent (unsteady). We are only concerned about steady flows in this work.

2.2 Streamlines

Field lines in flows are the curves generated by an initial starting location and some chosen property of a flow field. We focus on the field line of *streamlines*, which are curves that are tangent everywhere to the vectors in the flow. They can be thought of as the trajectories in the fluid from a point $p \in V$. A parameterization $s_p : \mathbb{R} \to D$ by time t such that

$$\frac{ds_p(t)}{dt} \times V(s_p(t)) = 0 \tag{2.1}$$

is a streamline that begins or is "seeded" from point p. If we let $s_p(t) = (s_p^1(t), s_p^2(t), ..., s_p^n(t))$ and $V(x) = (v_1(x), v_2(x), ..., v_n(x))$, where $x \in D$, then the above equation also tells us that

$$\frac{ds_p^1(t)}{v_1(s_p(t))} = \frac{ds_p^2(t)}{v_2(s_p(t))} = ... = \frac{ds_p^n(t)}{v_n(s_p(t))}. \tag{2.2}$$

Intuitively, this means that at each point along the streamline, the vector in the flow field and the tangent vector of the streamline are parallel. A streamline seeded at point p also has the restriction that $s_p(0) = p$. An example of streamlines are shown in Figure 2.1.

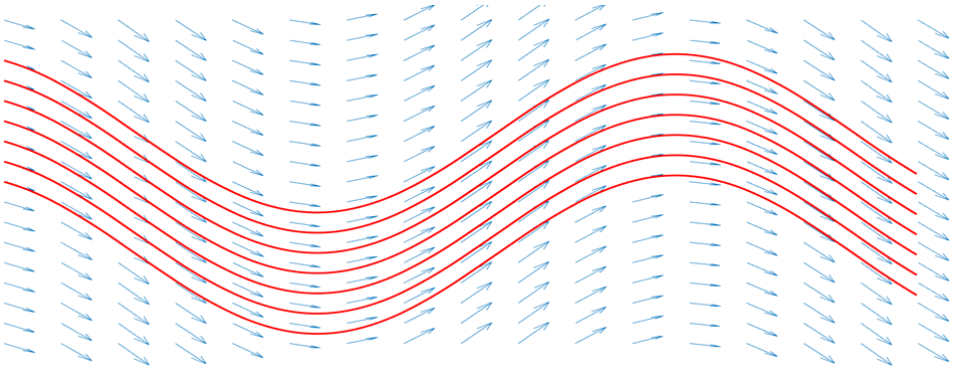


Figure 2.1: An example of streamlines seeded in a simple flow field.

Streamlines also follow the consistency condition. For any two times, t_1 and t_2, and point $q = s_p(t_1)$, we have $s_q(t_2) = s_p(t_1 + t_2)$ for all $p \in D$. The consistency condition means that a streamline seeded from a point that is contained in another streamline will result in two streamlines that follow the exact same trajectory. This prevents two different streamlines from intersecting each other at a single point without all of the other points in the trajectories identical.

2.3 Stream surfaces

Similarly to streamlines, *stream surfaces* are surfaces that are tangent everywhere to the flow. Stream surfaces can be generated by integrating a curve through the flow field. A stream surface S that is created through integrating curve c through the flow is said to be "seeded" from curve c. Additionally, c is referred to as the "seeding curve" for stream surface S. Stream surfaces can also be thought of as the union of streamlines seeded at an infinite density along this seeding curve. Let the seeding curve be the function $c : [0, 1] \to D$. A stream surface can now be defined as function $S : \mathbb{R} \times [0, 1] \to D$ such that $S(x, y) = s_p(x)$ where $p = c(y)$. An example of a stream surface is shown in Figure 2.2.

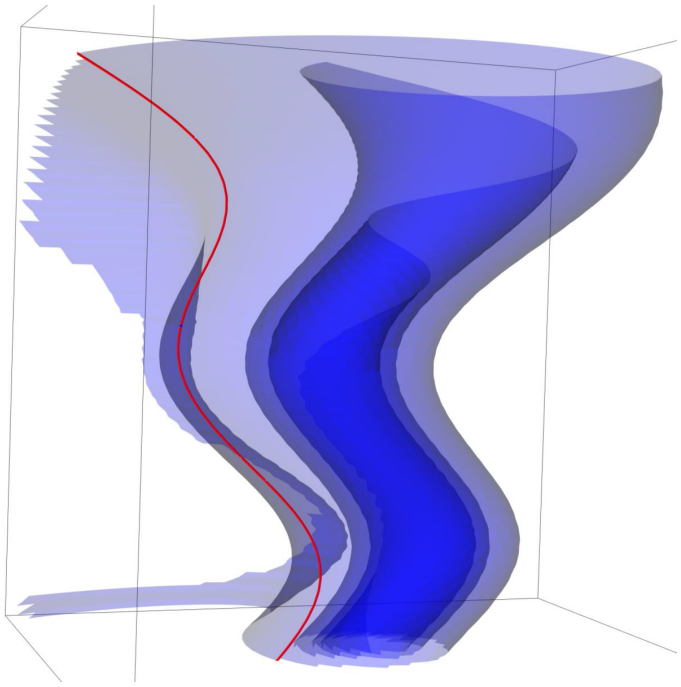


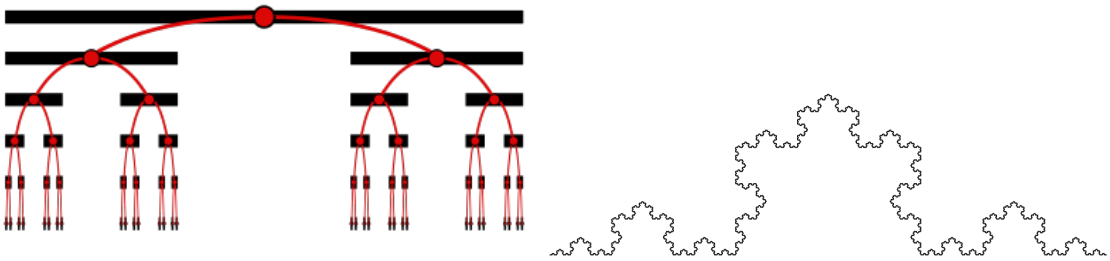
Figure 2.2: An example of a stream surface that represents a tornado. The seeding curve can be seen in red on the left.

An important distinction between stream surfaces and streamlines that makes stream surface visualization particularly difficult is that a single point defines an entire streamline, but does not define a stream surface. For instance, consider two line segments intersecting at a point. These will create two different stream surfaces that share a streamline at the intersection point of the curve. A region of a flow can be visualized by choosing points in that region and seeding streamlines from those points. This same technique cannot be used for stream surfaces. Stream surfaces require a seeding curve to be defined, which is arbitrary and independent of the flow itself.

2.4 Fractals

A fractal is a set that often displays self-similarity or detail at arbitrarily small scales. There is not a precise definition on what exactly make a set a fractal, but there tend to be certain features that fractals display. Strict definitions have excluded sets from being fractals that are typically thought of as fractals. Fractals tend to be defined through a recursive process, which tend to create a self-similarity property at various levels or scale. This means that as one would "zoom" in on a fractal, one could find that the fractal contains smaller versions of itself. Additionally, fractals tend to not be able to be described in traditional geometric terms.

Two examples of fractals are the "middle third Cantor set" and the "Koch Curve". The middle third Cantor set is displayed in Figure 2.3(a). Note that the middle third Cantor set does not include each iteration of construction shown in the figure. The middle third Cantor set is the set as the stage of construction approaches infinity. The Koch curve is shown in Figure 2.3(b).



(a) Six stages of construction of the middle third Cantor set. Obtained from [32].

(b) Koch curve. Obtained from [33].

Figure 2.3: Two commonly studied fractals.

2.5 Fractal dimension

The dimension of a set is typically thought of as an integer. Due to the irregular behavior of fractals, it often does not make sense to assign a fractal with an integer dimension. Fractals can have an infinite length when measured as a one-dimensional object, yet zero area when measured as a two-dimensional object, such as the Koch curve shown in Figure 2.3(b). We can instead assign a *fractal dimension* to a fractal, which may not necessarily be an integer. A fractal dimension describes how a set fills a space. There are many possible formulations of the fractal dimension all with their own properties, such as the Hausdorff dimension and the box-counting dimension.

2.6 Hausdorff dimension

One formulation of fractal dimension is known as the *Hausdorff dimension*. Before we introduce the Hausdorff dimension, we first must define the *Hausdorff measure*.

The *diameter* of $U \subseteq \mathbb{R}^n$ where $U \neq \emptyset$ is defined as $|U| = \sup\{|x - y| : x, y \in U\}$. In other words, the diameter of a non-empty subset of the n-dimensional Euclidean space is the greatest possible distance between two points in the set. A *cover* of a

set U is a collection of sets $\{U_i\}$ such that $U \subseteq \bigcup U_i$. A *δ-cover* of a set U is a cover such that $|U_i| \leq \delta$.

The Hausdorff measure [11] of a set $U \subseteq \mathbb{R}^n$ is now

$$\mathcal{H}^s(U) = \lim_{\delta \to 0} \inf\{\sum_{i=1}^{\infty} |U_i|^s : \{U_i\} \text{ is a } \delta\text{-cover of } U\}. \tag{2.3}$$

It can be seen that $\mathcal{H}^s(U)$ is non-increasing as s increases. Furthermore, it can be shown that if $\mathcal{H}^s(U) < \infty$ then $\mathcal{H}^t(U) = 0$ for all $t > s$. It is therefore implied that there exists a value in which the Hausdorff measure "jumps" from ∞ to 0. This value of jump in continuity, $dim_H U$, is known as the Hausdorff dimension [11] of U. In mathematical notation,

$$dim_H U = \sup\{s : \mathcal{H}^s(U) = \infty\} = \inf\{s : \mathcal{H}^s(U) = 0\}. \tag{2.4}$$

As one would expect, a simple one-dimensional line has a Hausdorff dimension of one and a two-dimensional disk has a Hausdorff dimension of two. Fractal sets are able to have non-integer fractal dimension. For example, the middle third Cantor set has a Hausdorff dimension of $\log(2)/\log(3) \approx .631$ and the Koch Curve has a Hausdorff dimension of $\log(4)/\log(3) \approx 1.262$.

2.7 Box-counting dimension

A more straightforward formulation of fractal dimension is known as the *box-counting dimension* [11]. Consider grid cubes G_δ in $\mathbb{R}^n$ of the form $[\delta \cdot m_1, \delta \cdot (m_1 + 1)] \times [\delta \cdot m_2, \delta \cdot (m_2 + 1)] \times ... \times [\delta \cdot m_n, \delta \cdot (m_n + 1)]$ where each m_i is an integer and $\delta > 0$. Now let $N_\delta(U)$ be the number of grid cubes of G_δ that a set $U \subseteq \mathbb{R}^n$ intersects. If the limit exists, we can now solve for $dim_B U$, the box-counting dimension of U with

the following:

$$dim_B U = \lim_{\delta \to 0} \frac{\log(N_\delta(U))}{-\log(\delta)} \tag{2.5}$$

The box-counting dimension of a set $U \subseteq \mathbb{R}^n$ intuitively calculates the rate at which the number cubes needed to cover a set grows as the cube edge length decreases.

Despite the different formulations, there remains a connection between the Hausdorff dimension and the box-counting dimension. When compared with the Hausdorff dimension, we know that for all $U \subseteq \mathbb{R}^n$, $dim_H U \leq dim_B U$. Additionally, the box-counting dimension is typically computed more easily than the Hausdorff dimension.

2.8 Box-counting ratio

One way to estimate the box-counting dimension is to sample δ at various values and then use linear regression to estimate the limit. In our work, we use a previously proposed method of estimation by Khoury and Wenger [15] that only requires δ to be sampled at two values. For a set $U \subseteq \mathbb{R}^n$, the *box-counting ratio*, $dim_R U$, by calculating the following slope with grid cubes sizes of δ and $\delta/2$.

$$\begin{aligned} dim_R U &= -\frac{\log(N_\delta(U)) - \log(N_{\delta/2}(U))}{\log(\delta) - \log(\delta/2)} \\ &= \frac{\log(N_{\delta/2}(U)) - \log(N_\delta(U))}{\log(2)} \\ &= \log_2 \left(\frac{N_{\delta/2}(U)}{N_\delta(U)} \right) \end{aligned} \tag{2.6}$$

Conveniently, in addition to only needing the values $N_\delta(U)$ and $N_{\delta/2}(U)$ to compute the box-counting ratio, we only need to compute the grid intersections of U for the grid $G_{\delta/2}$. The grid cubes intersected in G_δ can be directly obtained the grid cubes intersected in the grid cubes intersected in $G_{\delta/2}$. If grid cube with coordinates $(m_1, m_2, ..., m_n)$ is intersected in $G_{\delta/2}$, then it must be that grid cube

$(\lfloor m_1/2 \rfloor, \lfloor m_2/2 \rfloor, ..., \lfloor m_n/2 \rfloor)$ is also intersected. An example calculation of the box-counting ratio is shown in Figure 2.4.

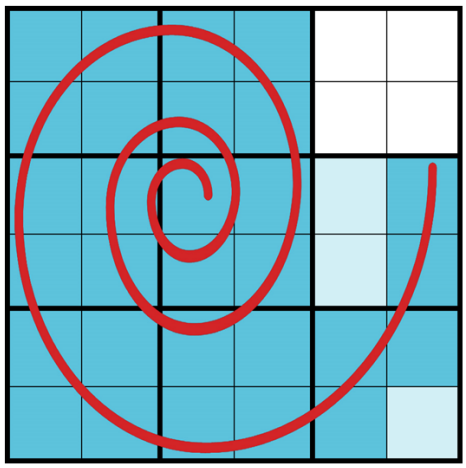


Figure 2.4: An example calculation of the box-counting ratio with grid cube sizes δ and $\delta/2$. See that the spiral intersects 8 boxes with edge length δ and intersects 29 boxes with edge length $\delta/2$. The box-counting ratio of this spiral is $\log_2\left(\frac{29}{8}\right) \approx 1.858$.

2.9 Fréchet distance

The *Fréchet distance* measures the similarity between two curves while also considering the order of the points in the curve. It is also sometimes referred to as the *dog walking distance*. Suppose that a man is walking on a side walk and his dog is walking on a different side walk next to him. The man and his dog are not allowed to move backwards, although they are allowed to move forward at different rates and possibly pause. The dog walking distance is the length of the shortest possible leash that would allow the dog and its owner to complete a walk from start to finish on each of their side walks, while remaining connected to each other by the leash.

Let $X : [0,1] \to \mathbb{R}^n$ and $Y : [0,1] \to \mathbb{R}^n$ be continuous functions representing curves. Also, let $a : [0,1] \to [0,1]$ and $b : [0,1] \to [0,1]$ be continuous, monotonically increasing functions where $a(0) = b(0) = 0$ and $a(1) = b(1) = 1$. The Fréchet distance between X and Y is

$$F(X,Y) = \min_{a,b} \max_{t \in [0,1]} \{d(X(a(t)), Y(b(t)))\}. \tag{2.7}$$

The Fréchet distance can be extended to discrete polygonal curves as well. Similarly to the continuous version, let $X = (x_1, x_2, ..., x_n)$ and $Y = (y_1, y_2, ..., y_m)$ be a sequence of points that define polygonal curves. Also, let $A = (a_1, a_2, ..., a_k)$ and $B = (b_1, b_2, ..., b_k)$ be monotonically increasing integer sequences such that no consecutive values have a difference greater than one, $a_1 = b_1 = 1$, $a_k = n$, and $b_k = m$. The discrete Fréchet distance between X and Y is

$$F'(X,Y) = \min_{a,b} \max_{t \in 1,2,...,k} \{d(x_{a_t}, y_{b_t})\}. \tag{2.8}$$

In the discrete version of the Fréchet distance, we only consider distances between the discrete points that define the polygonal curve, rather than all of the points in the curve. The discrete Fréchet distance is able to be computed in $O(mn)$ time with a straightforward dynamic programming approach. An example of a discrete Fréchet distance calculation is shown below in Figure 2.5.

We slightly modify the definition of the discrete Fréchet distance for our specific application. The discrete Fréchet distance is used in this work to compute the similarity of curves generated in a flow field. In order to not penalize curves propagated in the flow field for different lengths of time, we truncate the discrete Fréchet distance. This new definition modifies the sequences A and B to allow for either $a_k = n$ and $b_k = m'$, where $m' \leq m$ or $a_k = n'$ and $b_k = m$, where $n' \leq n$. This definition

allows for precisely one of the curves in the discrete Fréchet distance measurement to be truncated. The discrete Fréchet is then computed as usual once the sequence definitions have been altered. Let the truncated Fréchet distance of two polygonal curves, A and B, be denoted as $F_T(A, B)$.

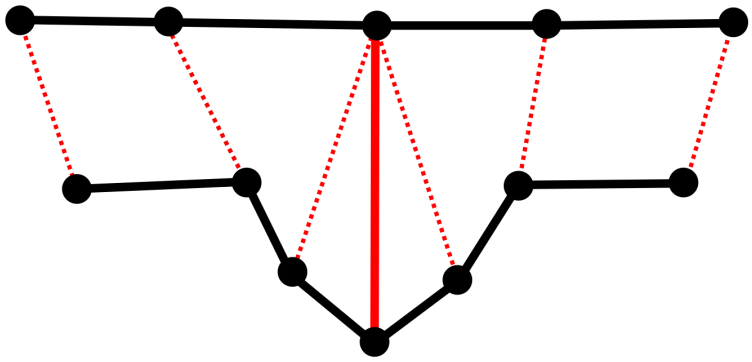


Figure 2.5: An example discrete Fréchet distance calculation. The discrete Fréchet distance between the two black polygonal curves is the length of the bold red line segment.

2.10 Intersection between ellipsoids and planes

Our algorithm to construct a stream surface seeding curve requires computing the intersection between an axis-aligned ellipsoid and a plane through the origin. Let the ellipsoid have semi-axes a, b, and c and be of the form

$$\frac{x_1^2}{a^2} + \frac{x_2^2}{b^2} + \frac{x_3^2}{c^2} = 1. \tag{2.9}$$

We wish to find the ellipse created by the intersection of some plane, Π, that contains the origin. We use the following results presented by Klein [16] to find this ellipse. Let the plane have the unit normal vector $n = (n_1, n_2, n_3)^T$ and be spanned by vectors $r = (r_1, r_2, r_3)^T$ and $s = (s_1, s_2, s_3)^T$. Let $D = diag(\frac{1}{a}, \frac{1}{b}, \frac{1}{c})$ be the diagonal matrix. Now, the form of the desired ellipse created by the intersection of the described ellipsoid and plane is

$$\langle t, u\rangle \begin{bmatrix} \langle Dr, Dr\rangle & \langle Dr, Ds\rangle \\ \langle Dr, Ds\rangle & \langle Ds, Ds\rangle \end{bmatrix} \langle t, u\rangle^T = 1. \tag{2.10}$$

If we pick vectors r and s in such a way that $\langle Dr, Ds\rangle = 0$, then the ellipse reduces to the form

$$t^2\langle Dr, Dr\rangle + u^2\langle Ds, Ds\rangle = 1. \tag{2.11}$$

Note that this ellipse still has the origin as its center and has semi-axes of length $\sqrt{\frac{1}{\langle Dr, Dr\rangle}}$ and $\sqrt{\frac{1}{\langle Ds, Ds\rangle}}$.

To choose the desired r and s to obtain the ellipse in the desired simplified form, we first start with an arbitrary vector $\hat{r}$ that is orthogonal to the vector n and we then obtain $\hat{s} = n \times \hat{r}$. We then rotate these vectors to find our final r and s vectors by the following equations:

$$r = \cos\omega\hat{r} + \sin\omega\hat{s} \tag{2.12}$$

$$s = -\sin\omega\hat{r} + \cos\omega\hat{s}. \tag{2.13}$$

See that

$$\langle Dr, Ds\rangle = 1/2 \cdot (\langle D\hat{s}, D\hat{s}\rangle - \langle D\hat{r}, D\hat{r}\rangle)\sin 2\omega + (\langle D\hat{r}, D\hat{s}\rangle)\cos 2\omega \tag{2.14}$$

This value is equal to zero precisely when

$$\frac{\langle D\hat{r}, D\hat{r}\rangle - \langle D\hat{s}, D\hat{s}\rangle}{2\langle D\hat{r}, D\hat{s}\rangle} = \cot 2\omega. \tag{2.15}$$

Therefore, we choose ω to be $1/2 \arctan\left(\frac{\langle D\hat{r}, D\hat{s}\rangle}{\langle D\hat{r}, D\hat{s}\rangle - \langle D\hat{s}, D\hat{s}\rangle}\right)$. In the case that $\langle D\hat{r}, D\hat{r}\rangle - \langle D\hat{s}, D\hat{s}\rangle = 0$, we are able to simply choose ω to be $1/4$. Finally, we calculate our final r and s vectors as using this ω value in the above equation. These r and s vectors are now vectors of the principal axes of the desired ellipse obtained by the intersection.

2.11 Axis-aligning ellipsoids

Let M be a $n \times n$ symmetric, positive definite matrix and see that the set $\{x | x^T M x = 1\}$ defines an n-dimensional ellipsoid centered at the origin. We look to provide a change of basis so that this ellipsoid becomes axis-aligned to simplify future calculations involving the ellipsoid.

Let $\{e_1, e_2, ..., e_n\}$ and $\{\lambda_1, \lambda_2, ..., \lambda_n\}$ be the unit eigenvectors and corresponding eigenvalues of the matrix M. Additionally, let U be an $n \times n$ matrix with e_i as the i^{th} column and let Σ be an $n \times n$ diagonal matrix such that $\Sigma_{ii} = \lambda_i$. See that $MU = U\Sigma$, from the definition of eigenvectors. Since M is symmetric and positive definite it is known that its eigenvectors are orthogonal, so $U^{-1} = U^T$ and we have $M = U\Sigma U^T$.

Therefore, if we use the matrix U^T as our change of basis in the form $y = U^T x$, we can show that

$$x^T M x = (Uy)^T (U\Sigma U^T)(Uy) = y^T (U^T U)\Sigma (U^T U) y = y^T \Sigma y. \tag{2.16}$$

The matrix Σ is diagonal, so the n-dimensional ellipsoid described by the set $\{x | x^T M x = 1\}$ is now axis-aligned under our new coordinate system.

2.12 Winding angle

The *winding angle* of a polygonal curve X defined by points $(x_1, x_2, ..., x_n)$ contained in a plane in 3D provides a signed angle measurement of how the curve has wrapped. Let $(v_1, v_2, ..., v_{n-1})$ be vectors between the points in curve X. For each $i = 1, ..., (n-2), (n-1)$, we have $v_i = (x_{i+1} - xi)$. Additionally, let this curve exist entirely in plane Π with unitized normal w. The winding angle is formally defined as

$$\Phi(X, w) = \sum_{k=1}^{n-1} \arcsin(w \cdot (v_i \times v_{i+1})) \tag{2.17}$$

Chapter 3: Applications of fractal dimension to flow field visualization

Flow fields are commonly visualized by streamlines. Streamlines are computationally inexpensive to generate and can provide the user with a clear picture of the behavior of a flow field. However, the placement of streamlines becomes problematic, especially in 3D flow fields. If a user is attempting to obtain an understanding of the entire flow field's behavior, then it is necessary to ensure that a sufficient number of streamlines are shown to capture how the flow behaves differently in different regions. On one hand, a high density seeding of streamlines will create a cluttered visualization. On the other hand, a low density seeding may miss important features. Thus, there is a need for algorithms to automatically choose streamlines for visualization. There have already been a number of algorithms and techniques in both 2D and 3D that automatically choose streamlines to show in a flow field visualization. Early methods of streamline placement focused on 2D flow fields. As data sets increased in size, more methods were developed for 3D flow fields which focus on streamline selection and feature identification.

3.1 Related work

An image-guided technique in 2D was introduced in [29] to uniformly place streamlines in the flow field by minimizing an energy function. The energy function considers the density of the image of the streamlines altered by a low-pass filter to achieve a uniform density. This method was extended to place streamlines on a surface in 3D in [19]. Another method to properly sample a 2D flow field is given in [22] and aims to generate long streamlines by seeding new streamlines at far distances from streamlines already seeded.

Entropy measurements are used in [34] and [17] to select streamlines which preserve as much information as possible about the flow field. The issue of occlusion in 3D is directly considered in [20] and a set of streamlines are chosen for a particular view that attempt to properly communicate the flow field, while preventing occlusion. In the case that a user has already has an region of interest to be visualized by a rake of streamlines, [21] uses similarity measurements to filter streamlines along a rake to reduce clutter while maintaining key features.

Critical points are extracted in [35] and various seeding strategies are used for different regions of the flow field. Flow field features such as vortices and divergence are described in [13] and then pattern matching is used to find these features in the flow field. Vortex behavior is specifically described and searched for in [25], [24], and [36]. Flow fields are topologically segmented in [18] in order to generate surfaces that separate the flow field into regions that have similar flow behavior.

3.2 Streamline complexity

In our approach to identify flow field features, we attempt to avoid solely looking for types of behavior such as vortices or streamlines with high curvature because such a search may exclude interesting behavior that does not meet such definitions. We instead consider the geometric complexity of streamlines in the flow field in order to identify regions of the flow field to display to a user. More specifically, we attempt to quantify the space-filling behavior of a streamline in order to determine its complexity. The use of this approach is motivated by the fact that the streamlines of many of the features flow fields that are interesting to users (such as vortices and turbulence) tend to exhibit space-filling behavior. If streamlines are found that fill a space densely, then potentially interesting regions of the flow field can be identified. We use the previously described box-counting ratio [15] in order to calculate the complexity of a streamline. The box-counting ratio was first applied to flow fields and streamlines in [4] where the space-filling behavior of streamlines is analyzed at a variety of scales. The box-counting ratio provides us with a simple way quantification of the complexity of flow field regions. These complexity measurements can then be used to build a scalar grid, which can be used in an interactive visualization to allow the user to explore the flow field and find various features.

3.3 Local box-counting ratio

We introduce the notion of the *local box-counting ratio* to accurately capture and quantify the behavior of a streamline near a single point in the flow field. Let ζ be a streamline generated from a flow field and let G_p be a $w \times w \times w$ grid of cubes centered at point p such that each grid cube has edge length λ. The local box-counting ratio of

ζ at p is $dim_R(\zeta \cap G_p)$ where the edge lengths used in box-counting ration calculations are $\lambda/2$ and λ. In other words, the local box-counting ratio of a streamline at a point p is simply the box-counting ratio of a certain portion of a streamline that is within a region around the point p.

The local box-counting ratio is used to capture the behavior of a streamline at a specific point in the flow field. If the box-counting ratio of an entire streamline is calculated to determine the complexity of a point in the flow field, the measurement may be misleading due to how the streamline behaves away from the point of interest. An example of such an instance is shown in Figure 3.1. Complexity measurements should be much higher in the regions in which the streamline exhibits vortex like behavior and much lower complexity measurements in the regions in which the streamlines are part of a more laminar flow. If the local box-counting ratio was not used, the information on the changing complexity of the streamline would be retained.

Additionally, for increased stability in the local box-counting ratio, we require that the value $N_\lambda(\zeta \cap G_p)$ from the box-counting ratio formula be above a certain threshold c. If the threshold is not met, then the measurement is discarded. This threshold is implemented to help ensure that a large portion of the streamline is being considered. In particular, this threshold helps obtain stable and consistent measurements near the grid boundary.

3.4 Streamline complexity grid

The streamline complexity grid is a scalar grid whose values represent the complexity of the flow behavior at each point, as computed by the box-counting ratio. This grid is constructed on top of the original flow field and the resolution of the grid

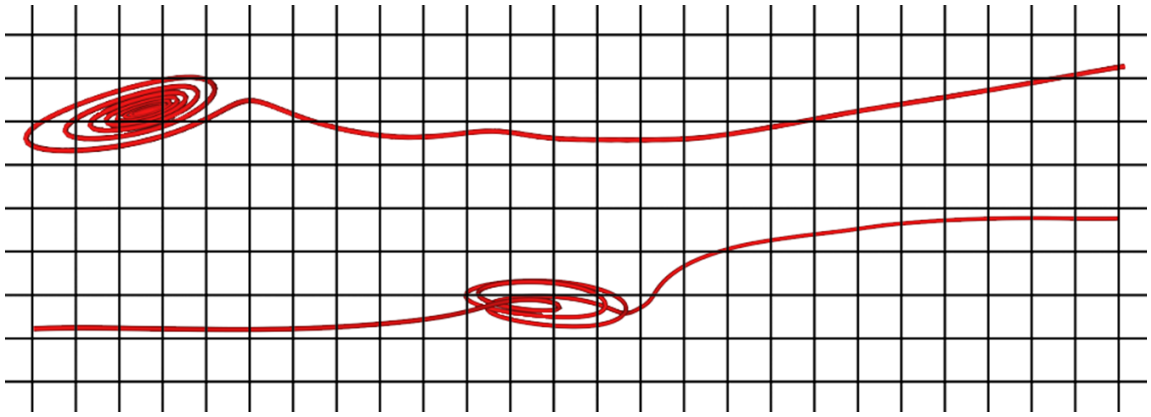


Figure 3.1: An example of streamlines that exhibit different levels of complexity along their lengths.

is a parameter chosen by the user before construction. Let the edge length of each grid cube in the streamline complexity grid be λ.

To create the streamline complexity grid it is first necessary to generate a dense enough sampling of streamlines to properly sample the vector field. We generate streamlines so that each grid voxel is intersected by at least 5 streamlines in the following way. For each voxel v_p associated with grid point p in the streamline complexity grid, seed a new streamline ζ through the center of v_p, if v_p does not already have 5 streamlines that intersect it. Now determine which voxels the streamline ζ intersects and associate ζ with a voxel if it has less than 5 streamlines that have previously intersected it. The limit of 5 streamlines per voxel is created to prevent an excessive amount of streamlines from being generated to reduce computation time and memory. We have empirically found that 5 streamlines per voxel provides a sufficient sampling.

Once this process is complete, the streamline complexity grid can now have scalar values assigned to each grid point. To calculate the scalar complexity value for grid point p, calculate the local box-counting ratio at point p of each streamline that was associated with v_p in the previous step. There will be a maximum of 5 streamlines per voxel. Then take the median of these local box-counting ratio measurements and denote the streamline responsible for generating that value as p_ζ. The final complexity value for the grid point p is then the average of this median value and the median values calculated in the same way of the 26 neighboring points. Denote this scalar complexity measurement value as $\phi(p)$.

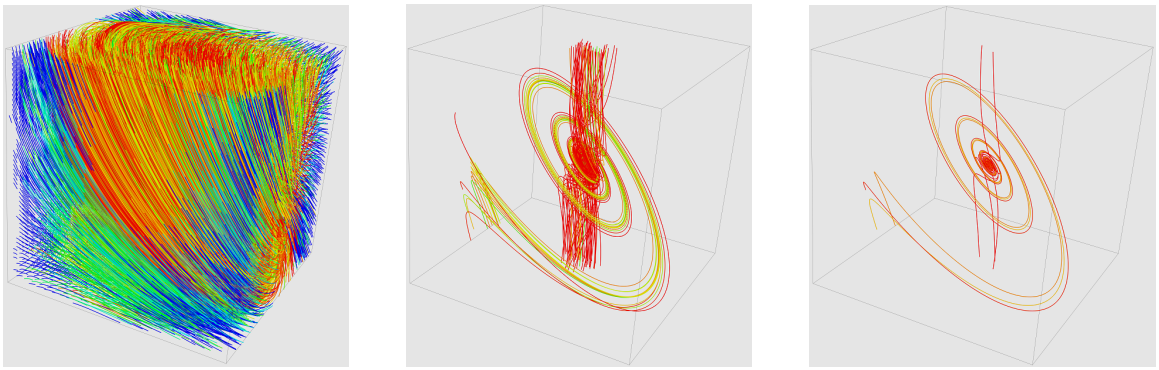
This streamline complexity grid provides a simple quantification of the regions of the flow field that have space-filling or complex behavior. It allows for the user to more easily filter through the flow field to find regions of interest. Additionally, previously known scalar field visualization techniques can be directly applied to the streamline complexity grid.

3.5 Visualization techniques

Here we outline how the streamline complexity measurements can be used to filter streamlines from the visualization. Additionally, we describe other visualization techniques that can be applied directly to the streamline complexity grid in order to allow the user to identify regions of high complexity. Note that these methods are designed to be interactive so that the user can examine the flow field data set in real-time. A stage of preprocessing is performed on the flow field data set and then the user is able to apply the following visualization techniques.

3.5.1 Streamline coloring

The complexity value of a streamline is defined to be the greatest local box-counting ratio measurement over all voxels that the streamline intersects. This decision is made again because a streamline can have significantly different behavior along its length. We want to label the streamline with its highest possible complexity value. The streamlines are colored by their complexity values with a rainbow color map to quickly indicate to a user which streamlines display space-filling behavior. Smooth streamlines with complexity measurements near 1 are blue, while streamlines that fill a space densely with a complexity measurement above 1.5 are red. This coloring scheme is shown in all of the following streamline images.



(a) (b) (c)

Figure 3.2: Example of the streamline filtering techniques applied to a synthetic data set. (a) The cluttered view of all 5000 streamlines generated from the data set. (b) 110 streamlines displayed after filtering by complexity measurements. (c) 8 streamlines displayed after filtering by local maximums. Increasing the complexity value to show less streamlines takes approximately 80 ms to calculate the new streamlines to be shown and then redraw the image.

3.5.2 Streamline filtering

To reduce clutter in the visualization, the user is able to remove streamlines from the visualization based on the measured complexity value of the streamlines. The user is able to select two values, a and b where $a < b$ and only display streamlines that had a measured complexity in between the two chosen values. The only streamlines that are shown when using this technique are the streamlines p_ζ such that $a < \phi(p) < b$. In other words, the user only sees the streamlines in the regions of flow that have a complexity value between the values a and b. If the user wishes to see smooth, laminar flow then the a and b parameters should be set near 1. If the user is attempting to find regions of the flow that is more turbulent, then the a and b parameters should be set above 1.4. An example of a synthetic flow field being filtered by complexity values is shown in Figure 3.3(b).

3.5.3 Local maximums of complexity measurements

A significant amount of streamlines can remain in the visualization if the streamlines are only filtered by the complexity measurements, especially if the grid cube edge lengths are relatively small. There can remain a significant amount of redundant information in the visualization if additional filtering steps are not taken. To further filter streamlines from the visualization, we also provide the option of only showing streamlines near local maximums of the streamline complexity grid in addition to filtering by complexity value. A streamline is only shown if its grid voxel is a local maximum in the streamline complexity grid. Local maximum filtering will only show streamline p_ζ if it meets the previously described filtering requirement and if for each q that neighbors p, $\phi(p) > \phi(q)$. This method allows for a single streamline to be a

representative of a high complexity region rather than showing each streamline in the region, which can clutter the visualization. Filtering by local maximums is shown in Figure 3.2(c).

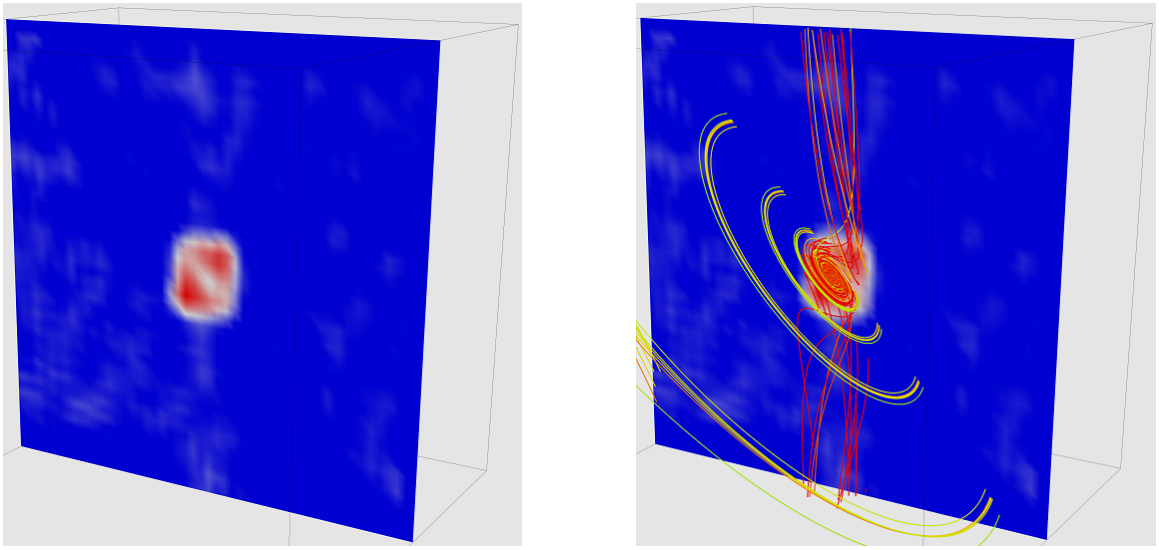
3.5.4 Complexity plane

A standard scalar grid visualization technique that can be directly applied to visualize the streamline complexity grid is viewing a heatmap of a cross section of the grid. A color gradient from blue to white to red is used and mapped to values in the range from 0 to 3, for each of the possible local box-counting ratio values. Low scalar complexity values are shown in blue colors and indicate to the user that a laminar flow is occurring in that region. Red colors denote flow with a high complexity and indicate to the user that a flow feature such as a vortex may be in that region. Our visualization allows for a plane with the above coloring scheme to be rendered and interacted with by the user. Additionally, either the x, y, or z coordinate of the plane is always fixed, to allow the user to easily manipulate the position of the plane.

The user is able to choose which coordinate of the plane is fixed and then move the plane along that axis. Once a region of interest is identified by the plane, the user is able to view the streamlines near the plane in order to see the actual behavior near that region of the flow field. To view the streamlines near the plane, the user chooses three parameters: a percentile P, a proportion for the top percentile a, and a proportion for the low percentile b. Once the parameters are chosen, we consider the voxels of the streamline complexity grid on the plane's current position. We then take the values of ϕ in the top P percentile and show proportion a of the streamlines in that top percentile. Similarly, we take the remaining streamlines not in the top P

percentile and show proportion b of those streamlines. This technique allows the user to view the behavior of high complexity regions of flow, while optionally obtaining context by viewing the streamlines in the surrounding low complexity region.

An example of this complexity plane is shown in Figure 3.3. Updating the coordinates of the plane takes approximately 250 ms to update the streamlines to show and to redraw the image.



(a) (b)

Figure 3.3: Example visualization using the colored plane. (a) The complexity plane indicating to the user that there is a region of high complexity in the center of the data set. (b) The streamlines shown from this high complexity region.

3.5.5 Isosurfaces

Isosurfaces can be used to highlight regions of high complexity to the user. Isosurfaces are the level sets of a 3D scalar field and are formally defined as $\{x|\phi(x) = \sigma\}$. By choosing a high σ value to compute the isosurface with, the isosurface will enclose

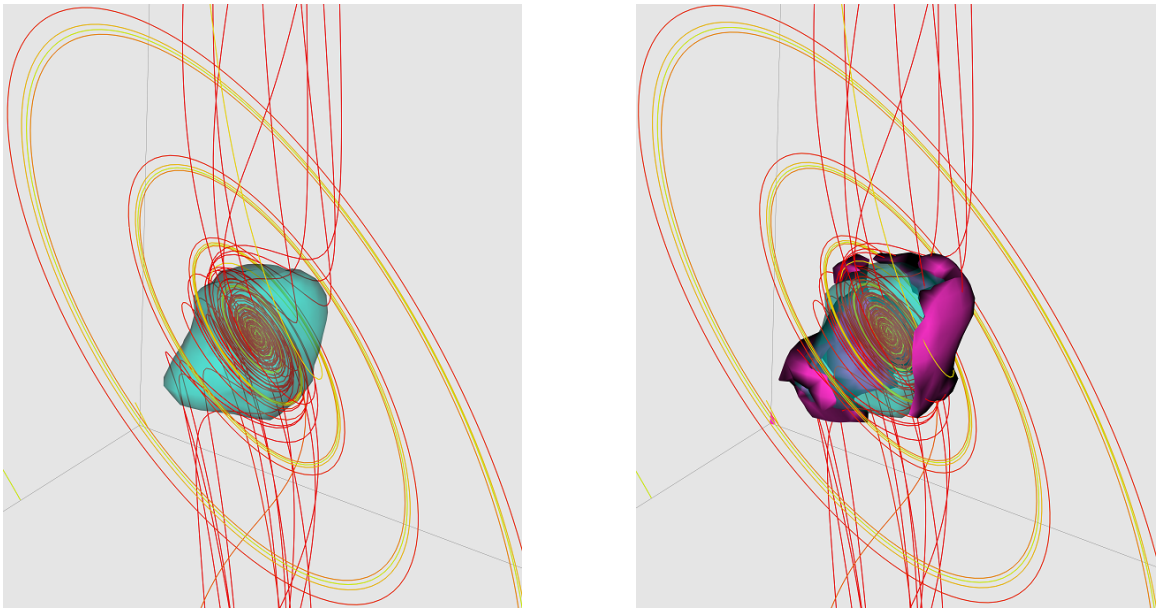
regions of the flow field that have high levels of complexity. This provides a simple way to quickly highlight all of the regions in the flow field that exhibit space-filling behavior or turbulence. In particular, isosurfaces at high complexity values often enclose the center of vortices in the flow field. An example of an isosurface visualization is shown in Figure 3.4(a).

3.5.6 Gradient magnitudes

The gradient magnitudes of the streamlines complexity grid can be calculated to create a new scalar grid, ϕ_g. The scalar $\phi_g(x)$ is given by $\|\nabla\phi(x)\|$. Another isosurface is used to visualize this scalar grid and can be used to highlight regions of the flow field in which the complexity changes quickly. When high isovalues are chosen to visualize the grid ϕ_g, the isosurfaces tend to highlight regions of isolated turbulence. The boundary of vortices are often captured by the isosurfaces of gradient magnitudes at high isovalues, as the center of the vortex has a high complexity value and its complexity quickly decreases as one moves away from the vortex. An example of such an isosurface is shown in Figure 3.4(b).

3.6 Results

This algorithm was implemented across two separate programs using C++ and The Visualization Toolkit (VTK). The first program preprocesses the flow field data set and creates the streamline complexity grid along with all of the streamlines to be rendered in a visualization. The second program only considers the streamline complexity grid along with the streamlines rendered in the visualization. Note that the second program does not consider the original flow field. Once the preprocessing step is complete, the original flow field data is no longer used in further steps of the



(a) (b)

Figure 3.4: Example visualization using the isosurfaces. (a) The center of a vortex being enclosed by an aqua isosurface with a high isovalue that is constructed from ϕ. (b) The boundary of the same vortex being enclosed by the purple isosurface that is constructed from ϕ_g.

visualization as all necessary information is already obtained. The data is preprocessed so that the user can ideally achieve near real-time interactive capabilities with the visualization.

3.6.1 Solar Plume data set

The Solar Plume data set ($126 \times 126 \times 512$) is a simulation of activity on the surface of the sun (provided by NCAR). A sample visualization of the Solar Plume is shown in Figure 3.5 with streamlines being seeded from the right side of the image and then traveling towards to left side. See that the Solar Plume exhibits mostly laminar flow towards the right side of the figure, while becoming more turbulent to the

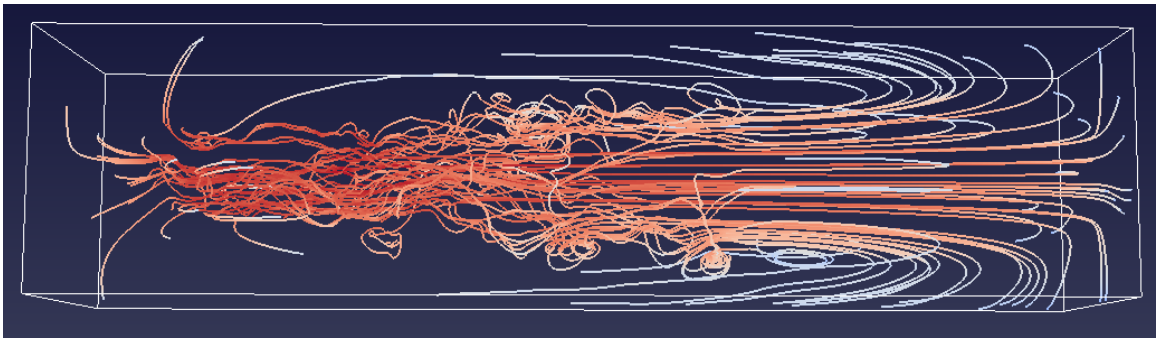
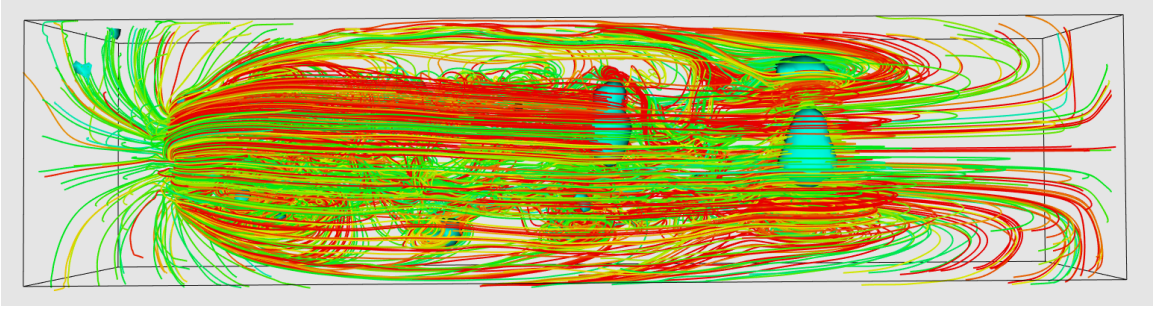


Figure 3.5: Streamlines sampled in the Solar Plume data set.

left. Additionally, several vortices become visible as the streamlines are integrated. The current arbitrary seeding is not suitable for exploration. Certain interesting features are visible from this seeding, but the cluttered streamlines makes exploration difficult. The following discussion will describe features found in the data set using our algorithm.

The algorithm used grid cubes with edge lengths of 3 and 1.5, a $12 \times 12 \times 12$ window for the local box-counting ratio, and a requirement of $N_3(\zeta \cap G_p)$ being at least 2. There are a total of 31581 streamlines generated in this visualization. It took 127 seconds to generate the streamlines and 94 seconds to perform the box-counting ratio calculations (including the time to compute the streamlines and grid intersections).

By setting the complexity cutoff to 1.25, we see the streamlines with a complexity value above 1.25. The isosurface of the streamline complexity grid also therefore has an isovalue of 1.25. The visualization with these described setting is shown in Figure 3.6(a). Despite filtering by the streamline complexity values, the visualize remains cluttered and it is still difficult to examine the regions of high complexity that a viewer may be interested in. We can further reduce the clutter in the visualization



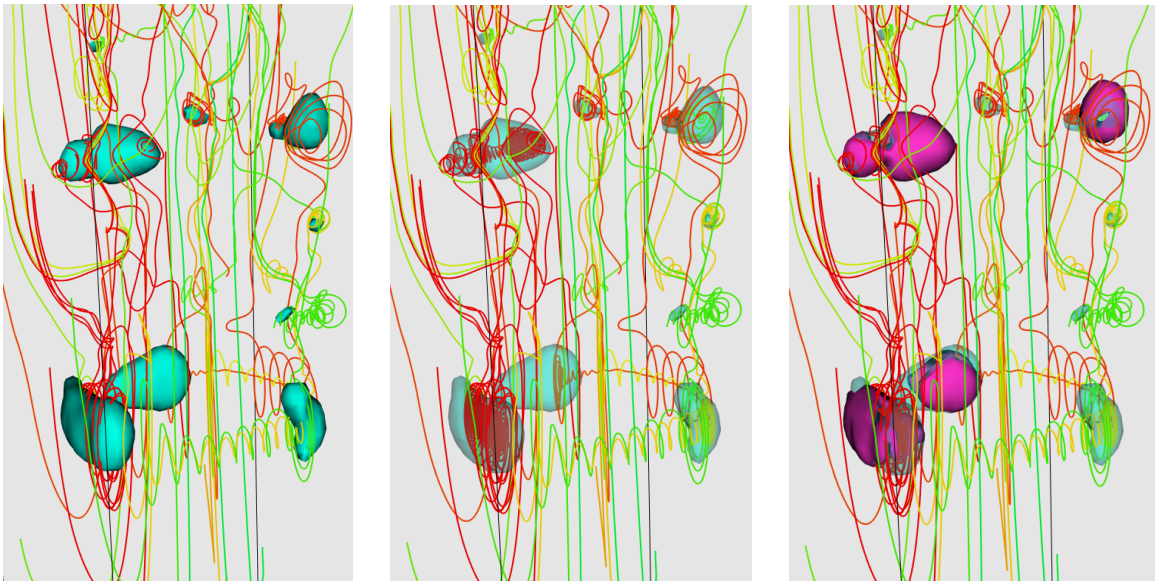
(a)

(b)

Figure 3.6: Visualization of the Solar Plume data set. (a) Streamlines generated from the flow field with a complexity value above 1.25 (a total of 439) (b) Streamlines generated from the flow field at the local maximums in the streamlines in the streamline complexity grid with complexity values about 1.25 (a total of 39).

by the previously described local maximum filtering. The streamlines at the local maximums in the streamline complexity grid being filtered by their complexity values is shown in Figure 3.6(b).

We can further examine the data set by zooming in on a region of the data set with several vortices that were highlighted by the isosurfaces. Figure 3.7 shows a region of the data set that contains several vortices. As the streamline complexity grid isosurfaces are made transparent in Figure 3.7(b), the viewer sees that the vortices are enclosed by the isosurfaces. Specifically, the vortices that exhibit a tight wrapping are the streamline that are highlighted are the vortices with a tight wrapping around



(a) (b) (c)

Figure 3.7: A region of the Solar Plume data set with several vortices. (a) Isosurfaces of the streamline complexity grid (in aqua) indicating regions that fill a space densely. (b) Isosurfaces made to be transparent to show the complex flow field behavior. (c) Isosurfaces of the gradient magnitude scalar grid (in purple) highlighting regions that change in complexity quickly.

the core. Although a wrapping behavior can be identified in the yellow streamline and the green streamline near the bottom of Figure 3.7(b), the vortices do not fill the the space densely. Therefore, those streamlines do not have as high of a local box-counting ratio as the red streamlines that are highlighted by the isosurface.

The isosurfaces from the gradient magnitude grid shown in Figure 3.7(c) also wrap around the isosurfaces as the isovalue is lowered, shown in Figure 3.7(b). This shows that these vortices are surrounded by lower complexity activity in the flow field, because this particular region of the flow field has a large gradient magnitude. The surrounding area can be further investigated by visualizing the scalar values of the streamlines complexity grid with the previously described heatmap.

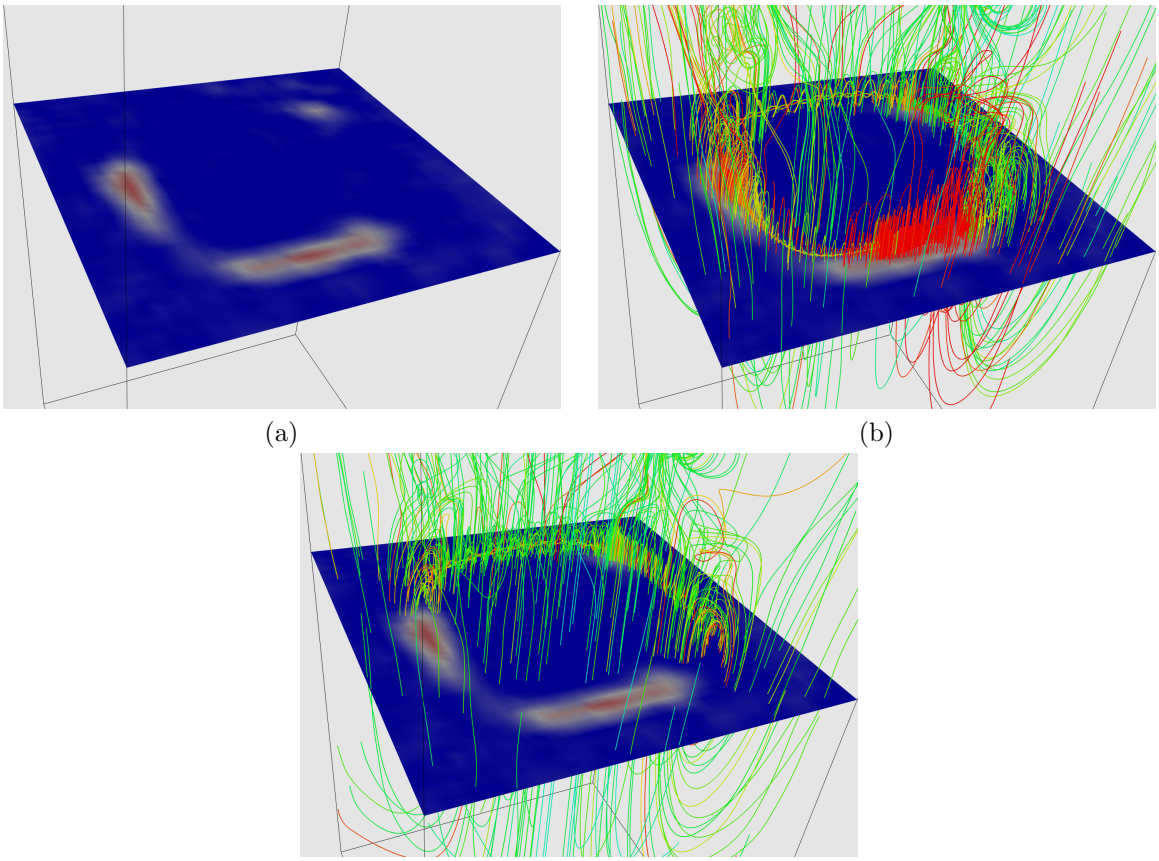


Figure 3.8: Utilizing the complexity plane in the Solar Plume data set. (a) The heat map of the complexities of the Solar Plume data set. High complexity regions are in red and low complexity regions are in blue. (b) Streamlines corresponding to the high complexity regions. (c) Streamlines corresponding to the low complexity regions.

A visualization using the complexity plane for the Solar Plume data set is shown in Figure 3.8. The coloring in Figure 3.8(a) first indicates which regions of the plane have streamlines with high complexity. The red and white regions are near streamlines with a high complexity while the blue regions (particularly towards the middle of the plane) are near streamlines with a low complexity. Figure 3.8(b) and Figure 3.8(c) show the streamlines near the high and low regions of complexity, respectively. Figure 3.8(b)

shows 12.5% of the streamlines that are in the top 50% of complexity measurements in the plane while Figure 3.8(c) shows 12.5% of the streamlines in the bottom 50%. Notice that the streamlines in Figure 3.8(b) are mainly focused in a ring with highly complex red streamlines. In Figure 3.8(c) the red streamlines are missing while more streamlines in the center of the data set. One can see from this visualization that this portion of the Solar Plume data set features several vortices that wrap around a more laminar region of flow.

3.6.2 Hurricane Isabel data set

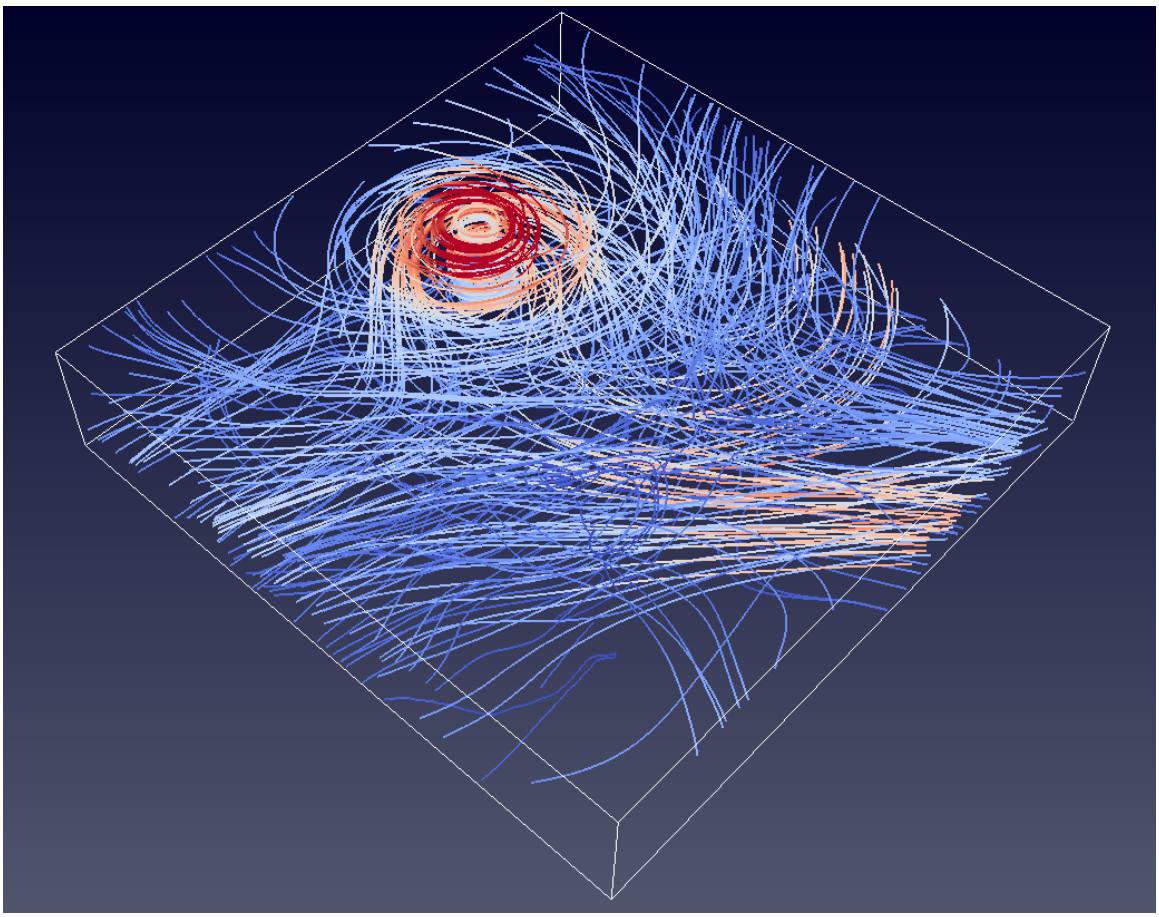


Figure 3.9: Streamlines sampled in the Hurricane Isabel data set.

The Hurricane Isabel data set ($500 \times 500 \times 100$) was obtained from the IEEE Visualization 2005 contest. A sample visualization of this data set is shown in Figure 3.9. A large vortex is able to be identified with an arbitrary seeding, however the visualization remains cluttered and there are vortices that are missed by this seeding. Unlike the previously shown Solar Plume data set, this data set has few larger vortices. We will use our algorithm to identify where these vortices are located.

The Hurricane Isabel data set was preprocessed using grid cubes with edge lengths of 4 and 2, a $16 \times 16 \times 16$ window for the local box-counting ratio, and a requirement of $N_4(\zeta \cap G_p)$ being at least 2. The algorithm generated a total of 12388 total streamlines for the visualization and computations and took 63 seconds to generate these streamlines. The box-counting ratio calculations and streamline intersection calculations were completed in 36 seconds.

Figure 3.10 shows streamlines from the Hurricane Isabel data set. Each individual figure displays the streamlines at different complexity cutoff values to demonstrate which features are shown as the complexity cutoff value increases. Figure 3.10(a) shows a dense sampling of streamlines. Some vortices in red can be identified, but the vortices are heavily occluded. The complexity value is increased in Figure 3.10(b) to show the vortices able to be identified in this data set. As the complexity value cutoff is continued to be increased in Figure 3.10(c) to 1.4, few dominant vortices are able to be identified. Figure 3.10(d) also has the complexity cutoff value of 1.4, but all streamlines are shown rather than only streamlines at the local maximums in the streamline complexity grid. In this particular case, viewing all of these streamlines does not clutter the visualization and instead adds more context more the viewer to observe the behavior around the vortices.

Further investigation using the heatmap shows the behavior of the lower complexity streamlines that are not part of the vortices. Figure 3.11 shows applications of the heat map to the Hurricane Isabel data set. In Figure 3.11 the regions of high complexity are highlighted in white and red. Streamlines are shown near these regions by displaying all the streamlines at the top 7.25% of the scalar values in the heat map. These streamlines are shown in Figure 3.11(b). To display streamlines around these vortices for increased contextual information, 1.5% of the streamlines in the bottom 92.5% of the scalar values in the heat map are shown in Figure 3.11(c). This allows a viewer to understand the behavior of the flow field between the vortices. In particular, a saddle point can be easily identified towards the center of the data set.

3.7 Discussion

Here we discuss how the previously explained parameters affect the visualization and explain the limitations of this approach.

3.7.1 Dependence on parameters

The parameters that have the most significant effect on the visualization and streamline complexity grid are λ (the grid edge length) and w (the window width used in the local box-counting ratio calculations).

There is a tradeoff between accuracy and computation time when choosing the λ parameter. When λ is chosen to be relatively small relative to the flow field dimensions, very fine features and small vortices can be identified using the local box-counting ratio. However, this causes a dramatic increase in computation time and storage space as more streamlines are needed to be generated to ensure a proper sampling of this grid with small edge lengths. A λ value is typically chosen so that

the streamline complexity grid has a slightly lower resolution than the original flow field. This is done to help capture features at the scale of the original data set, while maintaining a reasonable time for computations to complete.

The w parameter primarily influences the accuracy of the local box-counting ratio and the size of the features able to be identified. Ideally, we would use a very large w parameter to sample as much of the streamline as possible to increase the stability of the local box-counting ratio measurements and to decrease the effects of the grid alignment. However, if the w parameter is made to be too large, then we begin to sample too much of the streamline and the measurements could become distorted as we measure portions of the streamline away from a feature of interest. We typically choose a w parameter so that the size of the window used in the local box-counting ratio calculations is not significantly bigger than any of the features that we are trying capture.

3.7.2 Limitations

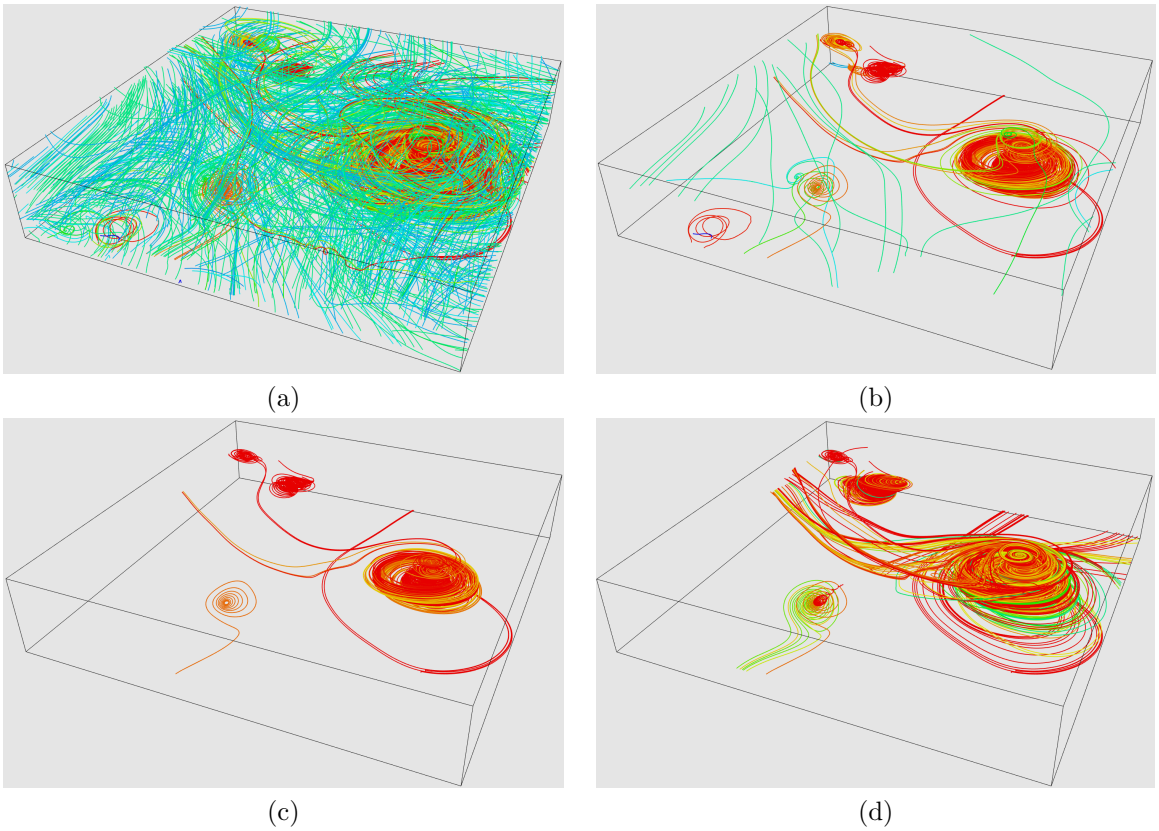
A limitation with this current approach is the alignment artifacts in the local box-counting ratio measurements due to the relatively small sample sizes of the streamlines. In the formulation of the box-counting dimension, the grid cube edge lengths approach zero. We should have very large sample sizes in our local box-counting ratio calculations. Noise and small errors due to the arbitrary grid alignment become more negligible as the sample size increases. To obtain a large sample size either the λ parameter should be set at a small value or the w parameter should be set at a large value. It was explained in the previous section on the dependence of parameters on

why both of these methods to obtain a large sample size are difficult. Therefore, it is difficult to obtain large sample sizes without suffering high computation times.

Additionally, even though the box-counting ratio is able to identify different types of turbulent behavior, we are not actually measuring anything about the vortices or turbulence themselves. This means that there is no guarantee that all of the vortices or turbulent regions in the data set are being accurately captures. Regions that have a high complexity value simply had a high local box-counting ratio. Due to previously discussed issues with grid alignment, it is possible that a vortex in the data set could be missed.

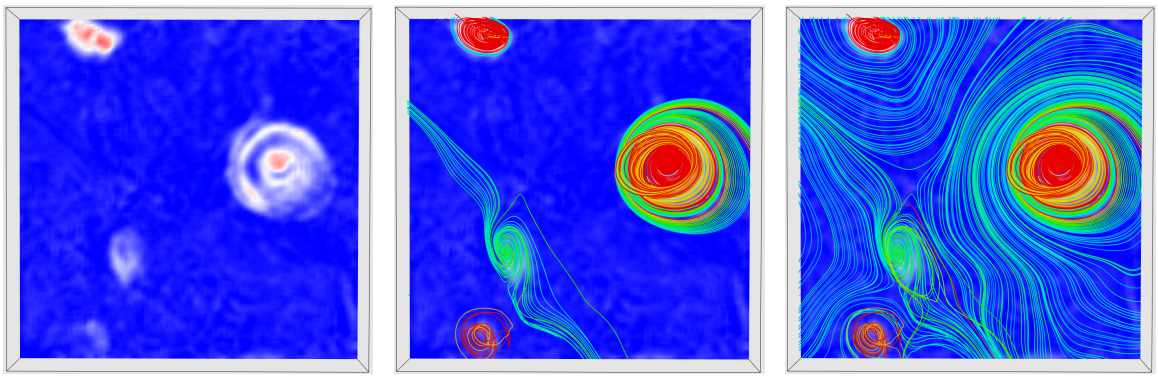
3.8 Conclusion

Here we present a method of identifying interesting flow field features by examining the geometry of streamlines seeded in the flow field. These complexity measurements are completed using the box-counting ratio measurements, which is an approximation of a formulation of a fractal dimension. Using the complexity measurements of the streamlines, we build a scalar grid over the original flow field that indicates which regions of the flow field exhibit complex flow behavior. The scalar grid is used in an interactive visualization where users can filter streamlines based on their complexity measurements or use standard visualization techniques on the scalar grid itself to quickly identify interesting regions of the flow field. We provide a examples and a discussion on how this visualization technique can be used on a Solar Plume flow field data set and a Hurricane Isabel flow field data set.



(a) (b) (c) (d)

Figure 3.10: Streamlines generated from the Hurricane Isabel data set filtered at different complexity values. (a) Streamlines shown at local maximums in the streamlines complexity grid with a complexity value over 1.1. (b) Streamlines shown at local maximums in the streamlines complexity grid with a complexity value over 1.23. (c) Streamlines shown at local maximums in the streamlines complexity grid with a complexity value over 1.4. (d) All streamlines with a complexity value over 1.4.



(a) (b) (c)

Figure 3.11: Utilization of the heatmap visualization for the Hurricane Isabel flow field. (a) Heatmap identifying multiple vortices in a slice of the flow field in red and white. (b) Streamlines near the high complexity regions in the heatmap seeded densely. (c) Both the streamlines shown in (b) and streamlines near low complexity regions seeded with a lower density.

Chapter 4: Automatic stream surface seeding curve generation based on vector similarities

While stream surfaces are effective at communicating the behavior of large regions of a flow field, they are not often used in practice due to the difficulty of generating them. Unlike a streamline, a single point in the flow field does not fully define a stream surface. Stream surfaces are generated by integrating a seeding curve through the flow field. This seeding curve does not depend on the flow field itself, so it is unclear how to generate the seeding curve.

Traditionally, to generate stream surfaces a user manually places a line segment in the flow field through trial and error and then uses the line segment as the seeding curve. This process is time consuming and often produces unsatisfying results. A line segment as a seeding curve arbitrarily groups streamlines together to make a stream surface and the resulting stream surfaces often do not provide a proper representation of the flow. A seeding curve that is not restricted to a line segment can produce a much more complex stream surfaces and a much more powerful visualization. Our research focuses on automatically generating seeding curves near points specified by the user in an efficient manner.

4.1 Related work

Here background is provided on both the integration of stream surfaces themselves and previous attempts of automatically generating stream surface seeding curves. For a more complete overview of stream surfaces, see the survey by Edmunds et al. [8].

4.1.1 Stream surface generation

Hultquist [14] describes an advancing front that is used to generate a stream surface. In this method, a seeding rake is advanced through the flow field and points are added or removed based on how far away points are from each other in the seeding rake. The points through the integration are then triangulated in order to create a surface that approximates a surface that is tangent everywhere to the flow field. Van Wijk [30] implicitly generates stream surfaces by assigning scalar values to some of the grid vertices and then constructing an isosurface that is an approximation of the surface that is tangent everywhere to the flow field. A method for creating stream surfaces in tetrahedral grids is described by Scheuermann et al. [26]. Garth et al. [12] describe a method to create more accurate stream surfaces around complex regions of the flow field. This method is particularly used to generate stream surfaces for vortices. The flow topology at critical points in the flow field is examined by Schneider et al. [27] to create more accurate surfaces near critical points.

4.1.2 Automatic stream surface seeding curve generation

Attempting to automatically generate seeding curves is a relatively newer research problem for stream surfaces. There has been significantly less work done in automatic seeding curve generation compared to other aspects of stream surfaces.

Edmunds et al. [9] create stream surfaces by examining how flow exits domain boundaries. Domains are defined in the flow field and then the vectors of the flow field on the boundaries are unitized and projected on to the domain boundary. The magnitude of this projected vector is taken and the boundary is assigned this magnitude as a scalar value at that point on the boundary. An isocontour is then generated on the boundary from these scalar values and the isocontour is used as a seeding curve from a stream surface. The vectors along seeding curves generated with this method all leave the boundary at the same angle.

Bartoň et al. [1] use the "stretch" of the seeding curve as it is integrated through the flow field to find an optimal seeding curve. The authors use a Taylor expansion of the seeding curve arc-length as it is integrated to determine how to seeding curve is stretched. They select a seeding curve which minimizes this stretch. A point in the flow field is chosen by the user and then a direction is chosen to advance in by solving for when the first two energy terms of the Taylor expansion are zero. However, this solution is only guaranteed to exist in flow fields that are divergence free. In case this solution does not exist, alternative methods of advancing the seeding curve to minimize stretching are also provided by the authors. The seeding curve is then altered by creating an initial stream surface from the seeding curve, refining this stream surface, and then tracing streamlines backwards from the new surface to solve for a new seeding curve.

Brambilla and Hauser [2] use streamline similarity measurements to find seeding curves. In this approach, streamlines are seeded densely on a patch in the flow field and the similarity of each pair of streamlines is measured. Each streamlines is then assigned a vector produced by a multidimensional scaling algorithm so that

the distances between the assigned vectors preserves the similarity measurements originally calculated for the streamlines. These vectors are then assigned to the point in the patch from which its streamline was generated. A seeding curve is then generated in this patch that minimizes the change in the vectors defined at each point in the patch. This ideally will create a stream surface consisting of similar streamlines.

The work by Peikert and Sadlo [23] examines the topology near critical points to create stream surfaces that can accurately express the behavior in these regions. Strategies for both creating a seeding curve near critical points as well as advancing the seeding curve are described.

Clustering is used by Edmunds et al. [7] to find starting locations for stream surface seeding curves. Vectors are clustered together based on parameters given by the user. The parameters supplied by the user help determine which stream surfaces will be created in the final visualization. Once the starting locations are determined, a seeding curve is created by integrating a through the curl field to create a seeding curve that is orthogonal to the vectors in the flow field.

Esturo et al. [10] densely sample the flow field with stream ribbons to aid in the creation of seeding curves. Multiple stream ribbons are seeded methodically around grid points in the flow field and then the quality of the stream ribbons are evaluated based on parameters provided by the user, depending on the type of stream surfaces the user wishes to see. A small number of the seeding curves used for the stream ribbons are then joined together considering the quality measurements to create a single seeding curve for a stream surface that is also high quality. The authors intend for this single stream surface to communicate a large amount of global information about the flow field. This work is extended by Schulze et al. [28] to create more than

one stream surface. A similar method is used in this extension but when additional stream surfaces are added into the visualization, the authors consider the distance of the new stream surface being added to all other stream surfaces already added in the visualization. This is done to properly sample the visualization with stream surfaces.

4.2 Motivation for seeding curve generation based on vector similarity

There are an infinite number of stream surfaces that intersect a single point in a flow field as well as an infinite number of stream surfaces that intersect a single streamline in a flow field. The large number of choices for a stream surface make it difficult to choose a surface to show to a user who wishes to see the behavior around a point in the flow field. Our approach seeks to generate a unique seeding curve from each point in the flow field. Unlike some previous methods that require minimal user interaction when choosing the starting location for seeding curves, our method requires the user to choose the point at which the seeding curve will begin at.

We consider two properties to create a seeding curve from point p in flow field f that "naturally" expresses features in the data.

1. The seeding curve remains in the plane that contains point p and is orthogonal to $f(p)$.

2. The vectors in f change as little as possible along the seeding curve.

Property 1 is established to both help prevent the creation of degenerate surfaces. If a seeding curve is generated that is nearly tangent to the flow, the stream surface that is created by this seeding curve will collapse into a degenerate surface. Instead,

it is required that the seeding curve be created orthogonal to the $f(p)$ to help prevent degeneracies near the point of interest. Additionally, this property is established to limit the search space of possible seeding curves. An infinite number of seeding curves can generate the same stream surface. We choose the seeding curve that comes from the intersection between the described plane and the stream surface.

Property 2 is established to help create a "natural" stream surface. While the seeding curve is restricted to the plane, the second property chooses a direction in the plane to advance the seeding curve. We intend to create stream surfaces that express features from the flow field. It is reasonable to expect that the vectors are similar along a seeding curve that creates a stream surface for a particular feature. The seeding curve is always advanced from a point q_1 to point q_2 such that the change from $f(q_1)$ to $f(q_2)$ is minimized considering the Jacobian matrix $J(q_1)$.

4.3 Algorithm for automatic seeding curve generation

Here the steps the algorithm takes to advance the seeding curve are described, from choosing the next direction, advancing, and terminating.

4.3.1 Choosing the next direction to advance the seeding curve

Let p be a point in the flow field, v be the vector at $f(p)$, and Π_p be the plane that contains p with v as its normal. We seek to find a unit vector e that is contained in Π_p that the seeding curve can be advanced in from p such that the change from v to the new point in the flow field is minimized. The Jacobian matrix is used to complete this. Let $J(p)$ be the Jacobian matrix of the flow field at point p. Now see

that the unit vector e that is contained in Π_p that minimizes $\langle (J(p))e, (J(p))e \rangle$ is the unit vector that is desired.

Let $M = (J(p))^T(J(p))$ and see that $\langle (J(p))e, (J(p))e \rangle = e^T M e$. The matrix M is positive definite, so the set $\{x | x^T M x = 1\}$ defines an ellipsoid. See that the unit vector described in the above minimization, e, is also the largest vector contained in Π_p that is also in $\{x | x^T M x = 1\}$. We will solve this minimization by searching for this largest vector in the 2D ellipse obtained by intersecting the plane Π_p and the set $\{x | x^T M x = 1\}$. As an overview of how this will be completed, we first axis-align the ellipsoid so that we can use results described in Section 2.10. Recall that the results in Section 2.10 can only be used for axis aligned ellipsoids. Once our ellipsoid is axis-aligned, we use the the results described in Section 2.10 to find the direction to advance in.

The result described in Section 2.11 is used to axis-align this ellipsoid to simplify the ellipsoid plane intersection calculations. From Section 2.11, let $M = U \Sigma U^T$, where matrix U^T provides the change of basis to axis-align the ellipsoid. In the implementation of this algorithm, the singular value decomposition (SVD) from VTK is used to solve for this decomposition.

It is also required to rotate the plane Π_p. Let $n = (U^T v)/\|(U^T v)\|$ be the normal vector to the plane Π that intersects the origin. Note that $\Sigma = diag(\frac{1}{a}, \frac{1}{b}, \frac{1}{c})$, so the results from Section 2.10 are now used to solve for the principal axes for the ellipsoid obtained by intersecting the set $\{x | x^T \Sigma x = 1\}$ and Π. Let the vectors r and s be the unitized vectors obtained by the process described in Section 2.10 using the matrix Σ and the plane Π. Define e' to be the vector r or s that corresponds with the smaller

of the values $r^T \Sigma r$ and $s^T \Sigma s$. See that $x = e'$ minimizes $x^T \Sigma x$ for vectors x in the plane Π.

Therefore, if we restore e' to our original coordinate system, the restored vector will minimize our original formulation. Let $e = Ue'$ and this vector is now the direction in the flow field that minimizes the change in v and stays in the plane Π_p. The seeding curve will be extended in direction e.

4.3.2 Advancing the seeding curve

The seeding curve starts at point p in the flow field f. Let λ be a predefined step length supplied by the user. The seeding curve is simultaneously extended a distance λ in a both a forward direction and backward direction at every iteration. Let the initial forward direction be labeled $e_1^+ = e$ and the initial backward direction be labeled $e_1^- = -e$, where e is the previously described optimal direction to advance in. Additionally, let the k^{th} point in the forward and backward seeding curve be denoted p_k^+ and p_k^-, respectively. Lastly, let $p_0^+ = p_0^- = p$.

At each step k, a new optimal direction e_k^σ is solved for both $\sigma = +$ and $\sigma = -$. See that both $-e_k^\sigma$ and e_k^σ provide a correct minimization. We will advance in the direction e_k^σ if $e_k^\sigma \cdot e_{(k-1)}^\sigma > 0$, and $-e_k^\sigma$ to prevent the seeding curve from flipping direction.

The seeding curve is able to change direction rapidly if the 2D ellipse obtained by the ellipsoid plane intersection is nearly circular. We put the follow measure in place to prevent the seeding curve from changing direction abruptly. If $e_k^\sigma \cdot e_{(k-1)}^\sigma < \cos(45°)$, we ignore the newly computed optimal direction and assign e_k^σ to be $e_{(k-1)}^\sigma$. The

seeding curve is able to change direction rapidly and appear jagged if this precaution is not in place.

Once these steps is complete, the k^{th} point is assigned to be $p_k^\sigma = \lambda e_k^\sigma + p_k^\sigma$, for both $\sigma = +$ and $\sigma = -$. This process is performed repeated until one of the termination conditions in the next section are met.

4.3.3 Terminating the seeding curve

Here we describe under what conditions the seeding curve generation can be terminated. As previously mentioned, the seeding curve is extended in both a forward and backward direction. The first three conditions described will only terminate a single direction of the seeding curve generation, while the final two will terminate the entire seeding curve generation. Each of these conditions are checked on each step of the seeding curve generation by examining point p_k^σ and e_k^σ.

1. p_k^σ is not in the domain D.

2. $f(p_k^\sigma)$ has a magnitude of zero.

3. For $i = k, (k-1), ..., (k-4)$ the directions e_i^σ were forced to be the same to prevent the seeding curve from being too jagged

The justification for including condition 1 is straightforward. Condition 2 is included because there is not a clear or defined way based on our method to advance the seeding curve. Also, if a seeding curve were extended through a region of the flow field with vectors of magnitude zero, then the stream surface may represent multiple unrelated features and be misleading. Condition 3 is included because this is in indication that a stream surface is being generated in an unstable region of the flow field.

By preventing the seeding curve from changing direction abruptly in this way, the seeding curve is moving in a direction closer to maximum change in vectors, which is also undesirable. If the seeding curve is forced to advance in such a way, then it may be an indication that this region of the flow field may not be represented well with a stream surface If condition 3 is found to be true and the seeding curve generation is terminated, then the last five points inserted to the seeding curve are removed, as this region will not be represented well by the seeding curve.

Conditions 4 and 5 consider the winding angle of the seeding curve, as described in Section 2.12. Let $X^+ = (p_1^+, p_2^+, ..., p_m^+)$ and $X^- = (p_1^-, p_2^-, ..., p_n^-)$ and let the seeding curve start at point p in flow field f. The winding of the entire seeding curve is then

$$\Theta = -\Phi(X^-, f(p)) + \Phi(X^+, f(p)) + \arcsin((f(p)/\|f(p)\|) \cdot (-e_1^- \times e_1^+))$$

where Φ is defined in Section 2.12.

If either of the following conditions are found to be true, then the entire seeding curve is terminated.

4. If the seeding curve completely wraps, or $\|\Theta > 360°\|$

5. If the winding angle of the forward or backward seeding curve exceeds 270°. In other words, if $\|\Phi(X^-, f(p))\| > 270°$ or $\|\Phi(X^+, f(p))\| > 270°$.

The winding angle is used to determine when the seeding curve has made a complete wrap. Condition 4 is created to prevent the seeding curve from wrapping around itself multiple times and occluding itself. Condition 5 is created to prevent a single direction of the seeding curve generation extending too far from its seed point. It is

undesirable to continue the seeding curve for too long as error will accumulate and the seeding curve may begin to diverge and create a stream surface that represents a different feature.

4.4 Multiple stream surfaces around a point

The previously described seeding curve generation algorithm is used in our visualization software to create several seeding curves in a single region of the flow field. A rake of points is defined from an initial point, p, provided by the user and then a seeding curve is generated from each point in the rake. This sampling of seeding curves is used to attempt to show all features near the point of interest provided by the user. The seeding curves and stream surfaces are then filtered using similarity measurements to prevent occlusion.

4.4.1 Generating the seeding curve rake

Let Π_p be the plane that contains point p and is orthogonal to the flow field vector $f(p)$ and let e be the direction of minimal change, obtained from the method described in Section 4.3.1. The rake is constructed in the direction of $e' = f(p) \times e$, the direction that is orthogonal to both $f(p)$ and e. This direction is chosen so that the rake stays in the plane Π_p and so that the seeding curves are sufficiently spaced from each other. It would be undesirable to construct several seeding curves that nearly overlap each other. Once this direction e' is chosen, a line segment with point p as a midpoint that is in e' is constructed with length γ. Points $p_1, p_2, ..., p_N$ are evenly placed along this line segment so that there are n points on each side of the original midpoint, for a total of $N = 2n + 1$ points.

A minor adjustment is made to the seeding curve generation algorithm for these points along the rake. The seeding curve generated from point p_k is no longer necessarily required to stay in the plane that contains point p_k and has normal $f(p_k)$. It is desired that all of the seeding curves lie in a single plane. The points $p_1, p_2, ..., p_N$ were place along a line segment that lies in the plane that contains p (or p_{n+1}) with normal $f(p)$. Therefore, all the seeding curves will be guaranteed to be in the same plane if $f(p)$ is always used as the normal for the plane the seeding curves are restricted to.

4.4.2 Filtering stream surfaces

While generating a number of stream surfaces around a single point will capture many features in the flow field, it can also very easily clutter the visualization. The stream surfaces are strategically filtered to prevent clutter while still retaining all of the different features captured by the stream surfaces. Similarity measurements are used to determine to cluster together stream surfaces that represent the same feature. A single stream surface is then shown from each cluster as a representative for that feature.

The similarity of two stream surfaces is approximated by calculating the similarity between their seeding curves and the streamlines that originate from the seeding curve starting locations. The filtering algorithm uses the truncated discrete Fréchet distance described in Section 2.9 to estimate the similarity between the curves. The seeding curve is chosen to be used in the similarity measurements because it provides an estimation of which region of the flow field the stream surface shows. The described streamline is chosen to be used in the similarity measurement because it provides an estimation of the type of flow field behavior the stream surface shows. By combining

the similarity measurements of the seeding curve and described streamline, we obtain an approximation of how similar the stream surfaces are.

Given n points $p_1, p_2, ..., p_n$ along the seeding rake, let $C^+ = (c_1^+, c_2^+, ..., c_n^+)$ and $C^- = (c_1^-, c_2^-, ..., c_n^-)$ denote the seeding curves in the positive and negative directions, respectively, originating from each point. Additionally, let $S^+ = (s_1^+, s_2^+, ..., s_n^+)$ and $S^- = (s_1^-, s_2^-, ..., s_n^-)$ denote the streamlines integrated in the positive and negative direction, respectively, originating from each point. The similarities between two of these curves are denoted as

$$F_c(i,j) = \max\{F_T(c_i^+, c_j^+), F_T(c_i^-, c_j^-)\} - d(p_i, p_j)$$

$$F_s(i,j) = \max\{F_T(s_i^+, s_j^+), F_T(s_i^-, s_j^-)\} - d(p_i, p_j)$$

where $d(p,q)$ is the Euclidean distance between two points, p and q, and $F_T(A,B)$ is the truncated Fréchet distance between two curves, A and B, as described in Section 2.9.

Finally, the similarity measurement estimate between stream surfaces generated from points p_i and p_j is

$$S(i,j) = \beta F_c(i,j) + (1-\beta)F_s(i,j)$$

where $\beta \in [0,1]$. In many of the visualizations, β is set to 0 as the streamlines are typically better at determining the similarity between stream surfaces than the seeding curves.

Considering these definitions, a similarity vector of n values is made such that the i^{th} value in the similarity vector describes the similarity of the i^{th} stream surface, with respect to its neighbors in the seeding rake. Specifically, for $i = 2, 3, ..., (n-1)$,

the i^{th} value in the similarity vector is assigned to be

$$\max\{S(i, i-1), S(i, i+1)\}$$

and simply $S(1, 2)$ and $S(n, n-1)$ for $i = 1$ and $i = n$, respectively. The mean and standard deviation of the n similarity measurements are then evaluated. The n stream surfaces are then broken into clusters by finding when two consecutive stream surfaces' similarity measurements exceed the mean similarity measurement by some factor of the standard deviation. The middle stream surface in each cluster is then shown in the final visualization as a representative of that flow field feature.

4.4.3 Generating the stream surfaces

Once the seeding curves have been generated, we use a straightforward approach to generate the stream surfaces. A streamline is seeded from sampled points along the seeding curve and a surface is made from these streamlines. Quads are used to create a surface from the streamlines if the streamlines are close enough to each other and have not diverged. However, once the seeding curves are generated any other stream surface seeding algorithm could be used.

4.5 Results

We display the stream surfaces generated by our algorithm on a number synthetic and simulated flow fields. Some of the data sets included are part of a time varying data set, although only a single time step of the data set is used. We also provide a case study of our seeding algorithm applied to make a visualization of a jet engine compressor data set. Domain experts note the effectiveness of our approach in generating stream surfaces that allow them to understand the stall mechanisms of a jet

engine compressor. Note that all of the locations to begin the seeding curves and parameters are provided by the user. We provide timings and discuss the performance of our algorithm.

4.5.1 Sample data sets

Tornado ($48 \times 48 \times 48$): This data set is a single time step of an unsteady flow field and was provided by Roger Crawfis [6]. It features a single vortex extending in the z direction.

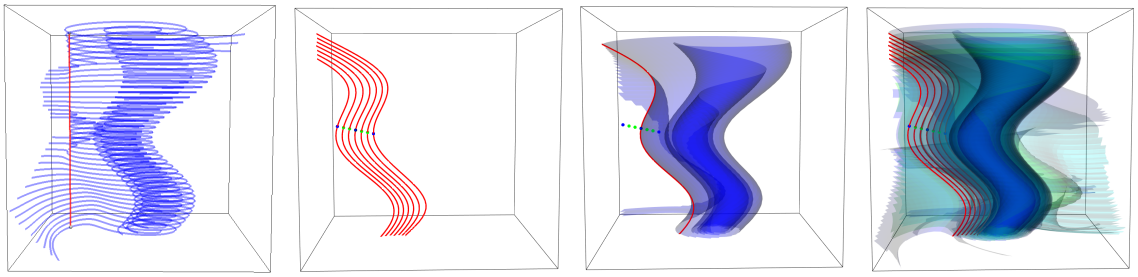


Figure 4.1: Visualizations of the Tornado flow field. (a) Streamlines seeded from a line segment as a seeding curve. (b) The seeding curves generated near a point of interest by our algorithm. (c) A single stream surface generated from a seeding curve generated by our algorithm. (d) Stream surfaces generated from all seeding curves generated by our algorithm show with opacity.

The vortex core of the Tornado data set is curved so the standard approach of using a line segment as a seeding curve will produce an unnatural looking stream surface. Streamlines that are seeded from a line segment are shown in Figure 4.1(a). Observe that the advancing front becomes heavily distorted as the streamlines are integrated due to the shape of the line segment seeding curve. Our algorithm generates the

seeding curves shown in Figure 4.1(b). A stream surface from one of these seeding curves is then shown in Figure 4.1(c). Notice that the streamlines that make up the stream surface seem to wrap in a way that shows the boundary of the tornado in the flow field. All of the stream surfaces from the seeding curves are shown in Figure 4.1(d).

Solar Plume ($126 \times 126 \times 512$): This data set was obtained from the National Center for Atmospheric Research (NCAR) and is part of a simulation of a solar plume occurring on the surface of the sun. A large portion of this data set is turbulent and cannot be visualized well with stream surfaces. Further analysis and visualizations of this data set are given in the previously presented work on streamlines in Section 3.6.1.

We focus on showing the more laminar regions of flow in the Solar Plume data set as the features in those regions are better suited towards visualizations with stream surfaces. The center of the Solar Plume data set contains flow that is moving in the upward z direction. The magnitudes of these vectors increase as one moves away from the center of the data set until the flow transitions into a vortex. This behavior is shown in Figure 4.2, which is a cross section of seeding curves and stream surfaces generated by our algorithm. Figure 4.2(a) shows all of the stream surfaces created from the seeding curves generated by our algorithm.

See that the stream surfaces towards the center are concentric and the stream surfaces towards the boundary properly show the vortex. Our seeding algorithm is able to clearly communicate this behavior because it creates seeding curves that minimizes the change vectors along the seeding curve. The seeding curves toward the center of the data set capture vectors that change minimally in magnitude, so

the curve remains a somewhat constant distance from the origin to create concentric seeding curves. Similarly, the seeding curves that capture the vortices wrap the boundary of the vortices to minimize change in vector magnitude. Figure 4.2(c) shows how the red seeding curve wraps the vortex boundaries in the visualization. Note that the green stream surface shown in Figure 4.2(c) is the same green stream surface shown in Figure 4.2(b).

The result of the stream surface filtering algorithm is shown in Figure 4.2(b). The filtering removes the redundant stream surfaces and preserves a stream surface for each feature shown by this set of stream surfaces. There is one stream surface that shows the center of the data set, one stream surface to show the transition to the vortex, and finally a single stream surface to show the vortex itself. Calculations from the filtering process will be shown to further explain these results.

Starting from the red seeding point in the center of the visualization going outward, the similarity measurements obtained are [8.96, 11.16, 30.04, 30.04, 27.62, 17.63, 17.63]. The mean of these values is 20.44 and the standard deviation is 8.19, which results in a cutoff for the clustering process of 28.63. Therefore, the stream surfaces shown in the final visualization are numbered 2, 4, and 6, where the stream surface that contains the red seeding point is surface number 1.

Flow Around a Cylinder ($192 \times 64 \times 48$): This flow field data set was originated from [3]. The version used in this work is a resampled version obtained from Tino Weinkauf [31]. This flow field is part of a single time step of an unsteady flow simulation of a fluid entering one side of a chamber, passing a square cylinder, and then leaving the other side of the chamber. The time step used in this work is the final time step of the visualization. The square cylinder in the data set is not visible

in the flow field data set, but it is located near the center of the data set before the vortices in Figure 4.3.

Results from our seeding curve algorithm are shown in Figure 4.3(a). Our algorithm generates seeding curves that are line segments extending across the data set. Notice that the stream surfaces produced by these seeding curves travel above or below the cylinder and do not tear. Additionally, the vortex that occurs behind the cylinder is able to be seen clearly from these stream surfaces.

The filtering process removes all by three of the stream surfaces and is displayed in Figure 4.3(b). See that the stream surfaces remaining show the behavior above and below the vortex and there is one stream surface remaining to show the vortex itself. Although many of the stream surfaces were removed, the distinct types of flow behavior shown by the stream surfaces are preserved and clutter is reduced.

Rayleigh-Bénard Heat Convection ($64 \times 64 \times 64$): The Rayleigh-Bénard Heat Convection flow field was provided by Bartoň et al. [1]. This data set is a simulation of a fluid that heated at a bottom boundary, causing the fluid to rise to the top and cool. Once the fluid reaches a top boundary and cools sufficiently, the fluid falls. The rising and falling of a fluid creates vortex like cells. This particular simulation contains four convection cells.

Our algorithm generates seeding curves that extend the length of the data set and create stream surfaces that accurately show the behavior of the convection cells. Figure 4.4(a) shows all of the stream surfaces generated with our method. Notice that the seeding curves appear to wrap the boundary of the vortices. Figure 4.4(b) shows a single stream surface produced by our algorithm with the direction of the flow rendered on the surface using LIC.

Francis Turbine ($100 \times 200 \times 74$): The Francis Turbine flow field was also provided by Bartoň et al. [1]. In Figure 4.5, air enters on the right side of the tube and travels down and left. A single stream surface seeded near the base is shown in Figure 4.5. This surface has been painted with LIC in order to show the direction of fluid flow. Notice in the magnified portion of the figure that the surface bends into a vortex when the air meets the divider. The combination of the streamlines and stream surface gives the viewer a clearer understanding of the flow in that region.

4.5.2 Case study: transonic jet engine simulation

Here we apply our technique to generate seeding curves to a transonic jet engine data set to create stream surfaces. In a work by Chen et al. [5], the authors seek to understand and visualize the stall mechanisms of a jet engine compressor. A complete overview of the work and previous visualization techniques can be viewed in [5].

During normal operating conditions of the jet engine compressor, a tip clearance vortex occurs at the leading edge of the blade tips near the compressor casing. As the compressor approaches a stall condition, these vortices breakdown and begin to oscillate. The current method the domain experts of [5] use to visualize these vortices is to manually place streamlines near the blade tips. These streamlines then reveal the structure of the vortices as they breakdown. This process of manually placing streamlines is very time consuming and can still produce an unsatisfying visualization. Streamlines lack contextual information and visualizing the behavior of flow structures can still be unclear even with a large number of streamlines.

We use our technique to automatically generate seeding curves that create stream surfaces that capture the vortex breakdown behavior. We select a time step of the jet

engine compressor data set in which the vortex breakdown is occurring and we place seeding points near where streamlines are traditionally placed by the domain experts to generate seeding curves. A visualization of this simulation is show in Figure 4.6. All 36 blades of the compressor are shown in Figure 4.6.a and the passage that we are examining the magnified in Figure 4.6.b. The vortex breakdown behavior that is of interest to the authors is magnified in Figure 4.6.c. The seeding curves that are used to generate these stream surfaces are magnified in Figure 4.6.d.

The stream surfaces generated by our method visualize the same behavior that the domain experts attempt to visualize with streamlines. Our method is able to provide more information to the viewer due to it being automatic. The user does not need to spend a considerable amount of time hand placing streamlines. Our seeding curves extend downstream in the data set to provide additional contextual information, that would be unreasonable to obtain with manually placed streamlines.

The domain experts commented that the additional information that the stream surfaces add to the visualization is helpful in understanding the vortex breakdown behavior and that it would be too time consuming to achieve the same results with manually placed streamlines. Viewing the stream surface geometry provides additional contextual information and allows the viewer to see how a region of the fluid behaves. The stream surface geometry also aided the experts in understanding the rotation and structures of the vortices. Additionally, the surface geometry let the experts see how the vortices detach during the breakdown and move downstream.

Viewing multiple stream surfaces in different colors such as the visualization in Figure 4.7 were also stated to be helpful by the domain experts. The domain experts commented that seeing multiple stream surfaces capture the same vortex allowed them

to further understand the size and rotation of the vortices. Automatically generating these surfaces provide much more descriptive visualizations than what was previously able to be done by manually placing streamlines.

As previously noted, our seeding curve generation algorithm seems to wrap the boundary of vortices and tends to create helpful stream surfaces to visualize vortex behavior. We use this property and generate additional stream surfaces downstream to better display the vortices in the data set. These stream surfaces are shown in Figure 4.8. The domain experts said that viewing multiple stream surfaces of different colors that represent a single vortex allowed them to verify that one end of the vortex is attaching to the casing while the other end is attaching to the blade. Creating such a visualization would be very time consuming to create with manually placed streamlines. Additionally, the domain experts said that they were not aware that the vortex in this region of the data set had what they referred to as a "tornado-like" structure. The stream surface geometry made it clear how exactly the vortex behaved.

4.5.3 Timings

The timings for our algorithm on the various data sets is shown in Table 4.1 and Table 4.2. The number of streamlines generated shown in Table 4.1 is directly proportional to the number of points along the stream surface seeding curve. For each point along the seeding curve, there are precisely one streamline generated in the forward integration direction and one streamline generated in the backward integration direction. The number of surface quads is then dependent on the number of streamlines and the resolution of the streamline generated. For less surface quads, a

Data Set	# of Surfaces	Average Number of	
		Streamlines	Surface Quads (10^3)
Tornado	7	459.57	16.55
Solar Plume	7	541.43	204.47
Flow Over Cylinder	9	229.11	125.81
Rayleigh-Bénard	10	322.60	312.58
Francis Turbine	3	730.00	330.99
Jet Engine (Figure 4.7)	7	549.43	1045.24
Jet Engine (Figure 4.8)	7	288.57	563.36

Table 4.1: Information about the number of streamlines and surface quads generated for each stream surface.

lower resolution streamline is able to be used. Note the most expensive computations in Table 4.2. The actual generation of the seeding curve is only a fraction of the time required to generate streamlines and build surface connectivity.

4.6 Discussion

Here we discuss how this seeding curve generation compares with previously proposed methods and the limitations of our described method.

4.6.1 Comparisons

The flow field data sets in this paper are often used in other works on stream surface seeding curve generation. We will compare the stream surface visualizations produced by our algorithm with the visualizations produced in other works. Comparisons with the methods of Bartoň et al. [1] and Brambilla and Hauser [2] are strongly focused on because these methods are most similar to our method.

Data Set	Average Time to Compute (ms)		
	Seeding Curves	Streamlines	Surface Quads
Tornado	20.00	96.14	104.86
Solar Plume	46.86	602.71	1225.43
Flow Over Cylinder	22.44	354.22	732.67
Rayleigh-Bénard	59.20	828.00	1853.40
Francis Turbine	58.00	916.00	1802.00
Jet Engine (Figure 4.7)	210.57	6202.29	5721.00
Jet Engine (Figure 4.8)	162.86	3388.43	3254.57

Table 4.2: Information on the time taken to compute the stream surfaces.

Seeding curves are generated by Bartoň et al. [1] so that the change in arc-length of the curve is minimized as it is integrated through the flow field. Although the final seeding curves are refined in this method, the original seeding curves are generated through a local inspection. However, their primary method depends on divergence free flow, which is not required by our method.

The method presented by Brambilla and Hauser [2] examines the behavior of densely seeded streamlines in order to find streamlines to group together into a seeding curve. This approach is more global in comparison to our method, but still seeks to capture a single flow feature. Using more global information requires a dense sampling of streamlines, where our method only requires information immediately available from the vector field.

Tornado ($48\times48\times48$): A stream surface visualization generated by our algorithm of this data set is shown in Figure 4.1. Our visualization focuses on the vortex core of the tornado. The results that we obtain are most similar to the results shown in Figure 7 of [9]. Notice that both methods produce long, bending seeding curves that

show the boundary of the tornado. Other visualizations of the tornado data set are also shown in Figure 8 of [2] and Figure 5 of [10]. These seeding curves are shorter and do not show the behavior of a seeding curve along the length of the data set. An advantage of performing local optimizations only is that we are able to generate long seeding curves at a low cost.

Flow Around a Cylinder ($192 \times 64 \times 48$): The stream surface visualization that is produced by our algorithm is shown in Figure 4.3. We aim to show the behavior of the fluid traveling through the entire data set by placing the seeding curves on one of the data set. The seeding curves that we generate are essentially identical to the stream surfaces shown in Figure 10 of [2]. This method also generates seeding curves that appear to be line segments. Figure 6 of [28] samples the entire data set with shorter seeding curves. The use of many small seeding curves does not show how a large region of flow moves together. Additionally, by placing many stream surfaces close together, occlusion issues arise. Figure 11 of [1] also shows results on this data set. The seeding curves generated with this method are again short and do not provide a visualization for how a large region of the flow moves together. The stream surfaces here do not show the user what is throughout the entire flow field. Additionally, the stream surfaces of Figure 10 in [2] and Figure 11 in [1] do not capture the vortex in the data set.

Rayleigh-Bénard Heat Convection ($64 \times 64 \times 64$): The results from our algorithm are shown in Figure 4.4. Recall that our seeding curves extend the length of this data set. This exact convection data set was also used in [1] and their results are displayed in Figure 9 of [1]. The seeding curves generated by this method again are short and do not sufficiently cover the data set, unlike the seeding curves generated

by our method. A viewer that is unfamiliar with the data set may have difficulty seeing and understanding the global behavior of this convection data set.

Other convection data sets are used shown in Figure 15 of [7] and in Figure 7 of [28]. Notice that the seeding curves in these results better emphasize the vortex cores of the convection cells. These approaches are able to capture such behavior because they examine the entire domain.

Francis Turbine ($100 \times 200 \times 74$): The stream surfaces our algorithm generated for this data set are shown in Figure 4.5. A stream surface visualization of this data set is also shown in Figure 10 of [1]. The stream surface visualization in [1] is again limited by the length of the seeding curves. The seeding curves only create stream surfaces that show a small portion of the flow field, while our algorithm is able to create a single stream surface that covers nearly the entire data set. Additionally, our algorithm captures a vortex which is not captured in Figure 10 of [1].

4.6.2 Limitations

Our method uses solely local information when constructing seeding curves. The seeding curves typically describe the local behavior of the flow field accurately, when the seeding curves are well-behaved. Since we do not account for any global or non-local information, we cannot make any claims about how the stream surface will behave as the seeding curve is integrated. Other methods such as [10, 28, 1] account for non-local behavior in order to obtain stream surfaces that are better behaved away from the seeding curve. However, these global approaches are significantly more time consuming.

Our algorithm performs best when the 2D ellipse described in Section 4.3.1 is elongated. An elongated ellipse means that one direction is significantly better to advance in than all others. Our algorithm does not perform as well in areas in which the 2D ellipse is nearly circular.

Figure 4.9 provides an example of a seeding curve that does not have an elongated ellipse. The outermost seeding curve does not behave like the other seeding curves because the 2D ellipse obtained towards the end of the seeding curve is nearly circular and it is able to bend away. Unpredictable behavior can also occur in turbulent regions of the flow field. Seeding curves can behave unexpectedly while traveling through a chaotic or turbulent region and generate a stream surface that is unhelpful to visualize the region of flow. However, in either case it may be that stream surfaces are not suitable to visualize the region of flow with. A different visualization method may be more suitable to visualize a region with high turbulence.

4.7 Conclusion

Stream surfaces can provide rich visualizations for flow field data sets. They provide an intuitive visualization for how the flow in a large region is behaving. The traditional method of hand placing seeding curves is time consuming and can still create unsatisfying visualizations. Our work provides an method to automatically create seeding curves from a single point to create informative stream surfaces. We describe properties of a desirable seeding curve and then create an algorithm that creates curves with these properties. Additionally, we provide a method to filter stream surfaces along a rake based on similarity measurements. The case study of

our algorithm applied to a transonic jet engine simulation further demonstrates the effectiveness of our approach in a real application. The domain experts emphasize that our algorithm produces stream surfaces that provide clear visualizations for phenomena being studied in the jet engine. The effectiveness of our method is demonstrated by the visualizations created from the discussed flow field data sets. We are able to easily select points of interest in the flow field and then create informative stream surfaces.

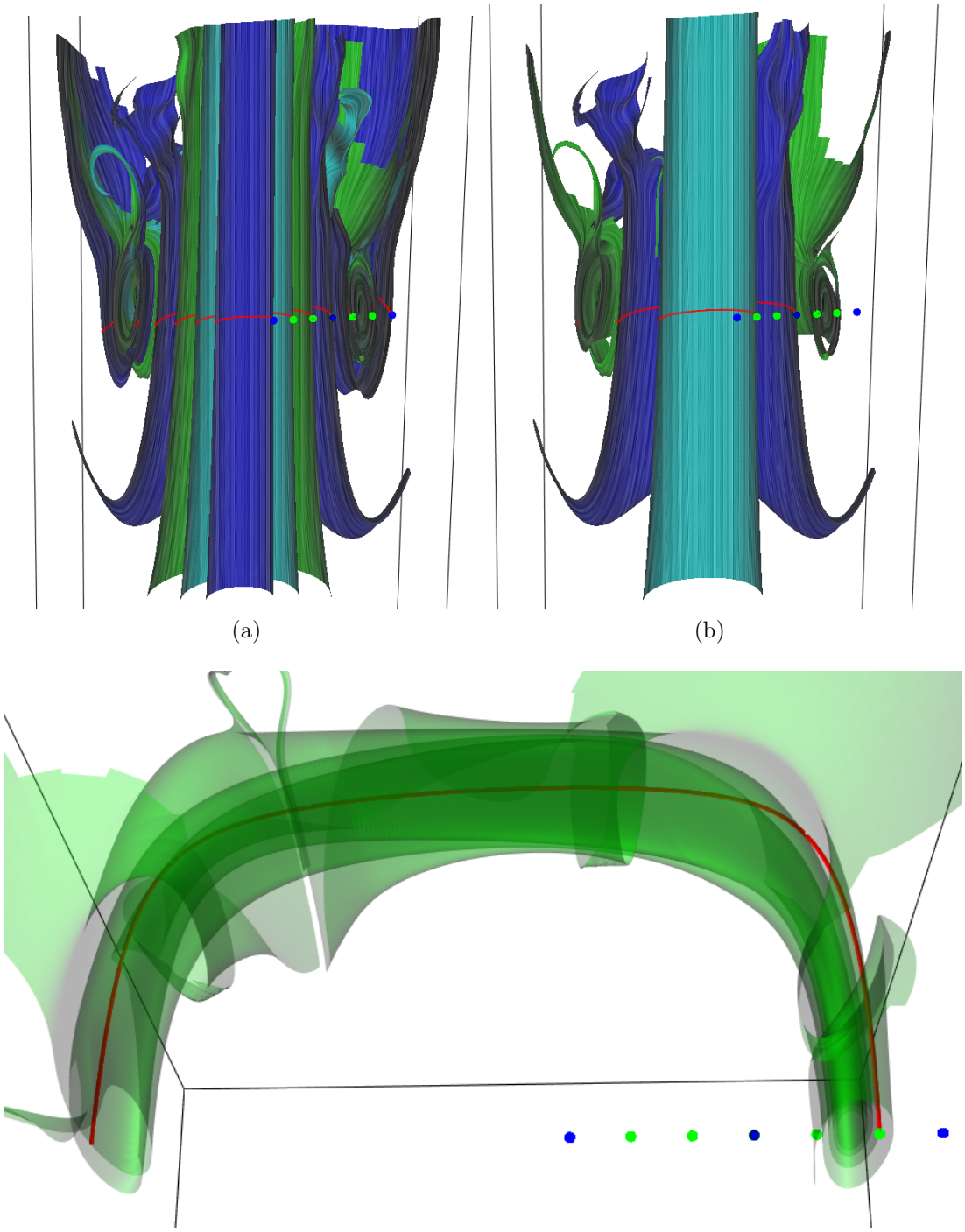
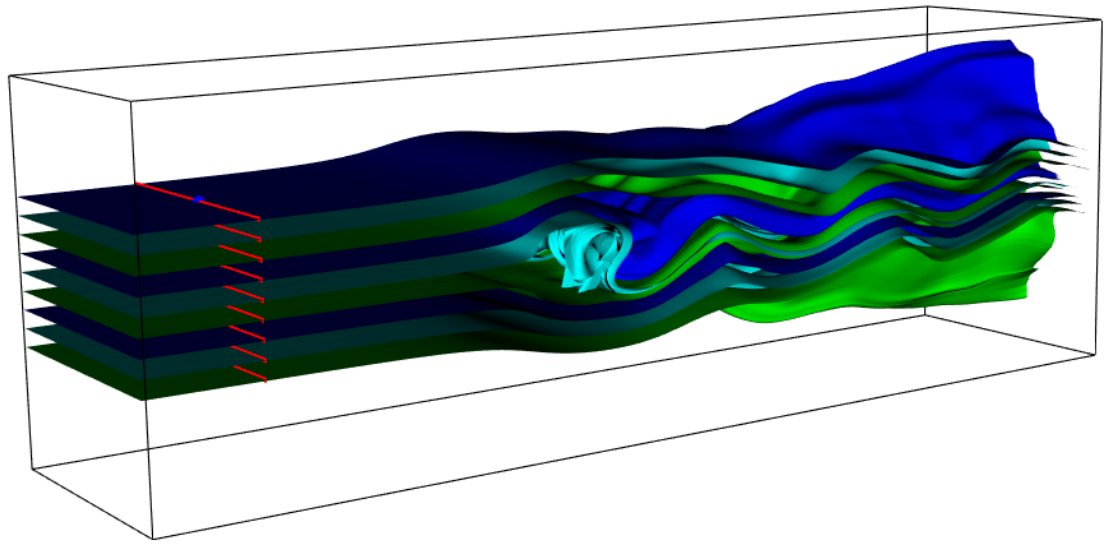
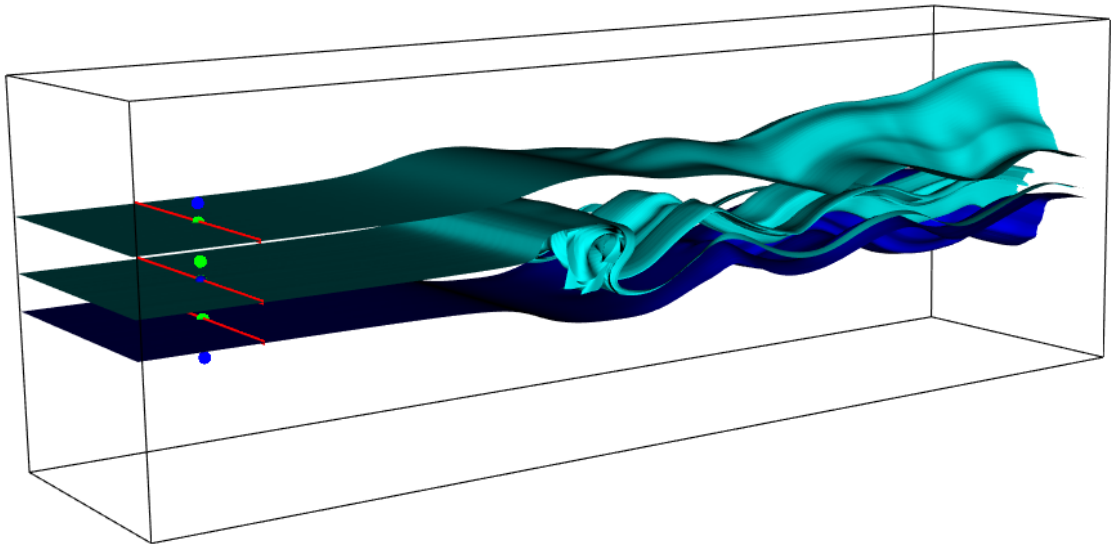


Figure 4.2: Stream surface visualizations of the Solar Plume flow field. (a) Stream surfaces from all seeding curves generated shown with Line Integral Convolution (LIC). (b) Stream surfaces filtered from the visualization to reduce clutter.

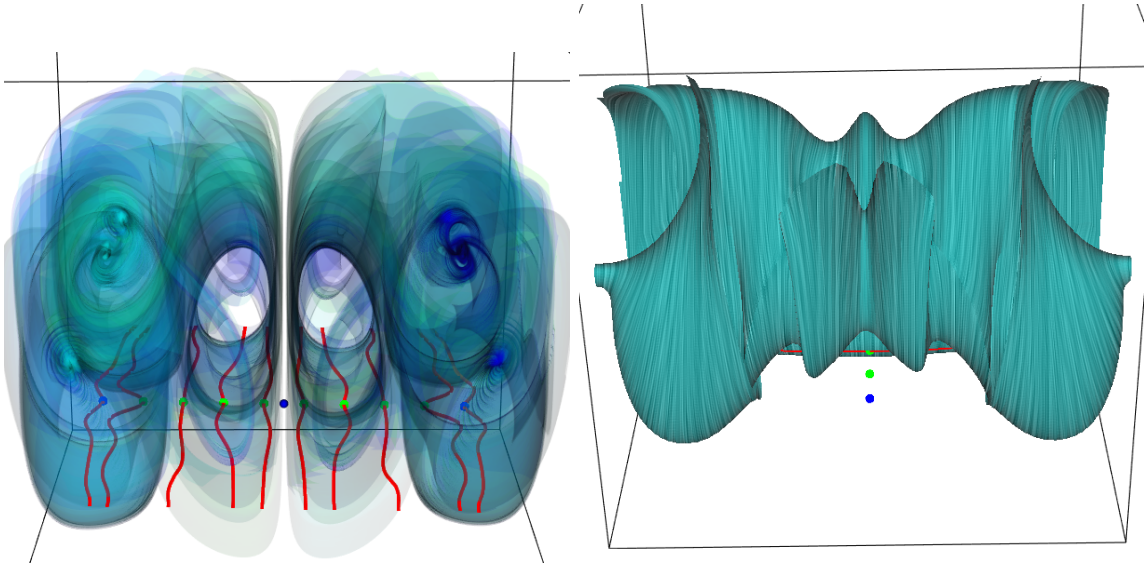


(a)



(b)

Figure 4.3: Stream surface visualizations of the Flow Around a Cylinder flow field. (a) Stream surfaces from all seeding curves generated. (b) Stream surfaces filtered from the visualization to reduce clutter.



(a) (b)

Figure 4.4: Stream surface visualizations of the Rayleigh-Bénard Heat Convection flow field. (a) Stream surfaces from all seeding curves generated show with opacity. (b) Side view of a single stream surface shown with LIC.

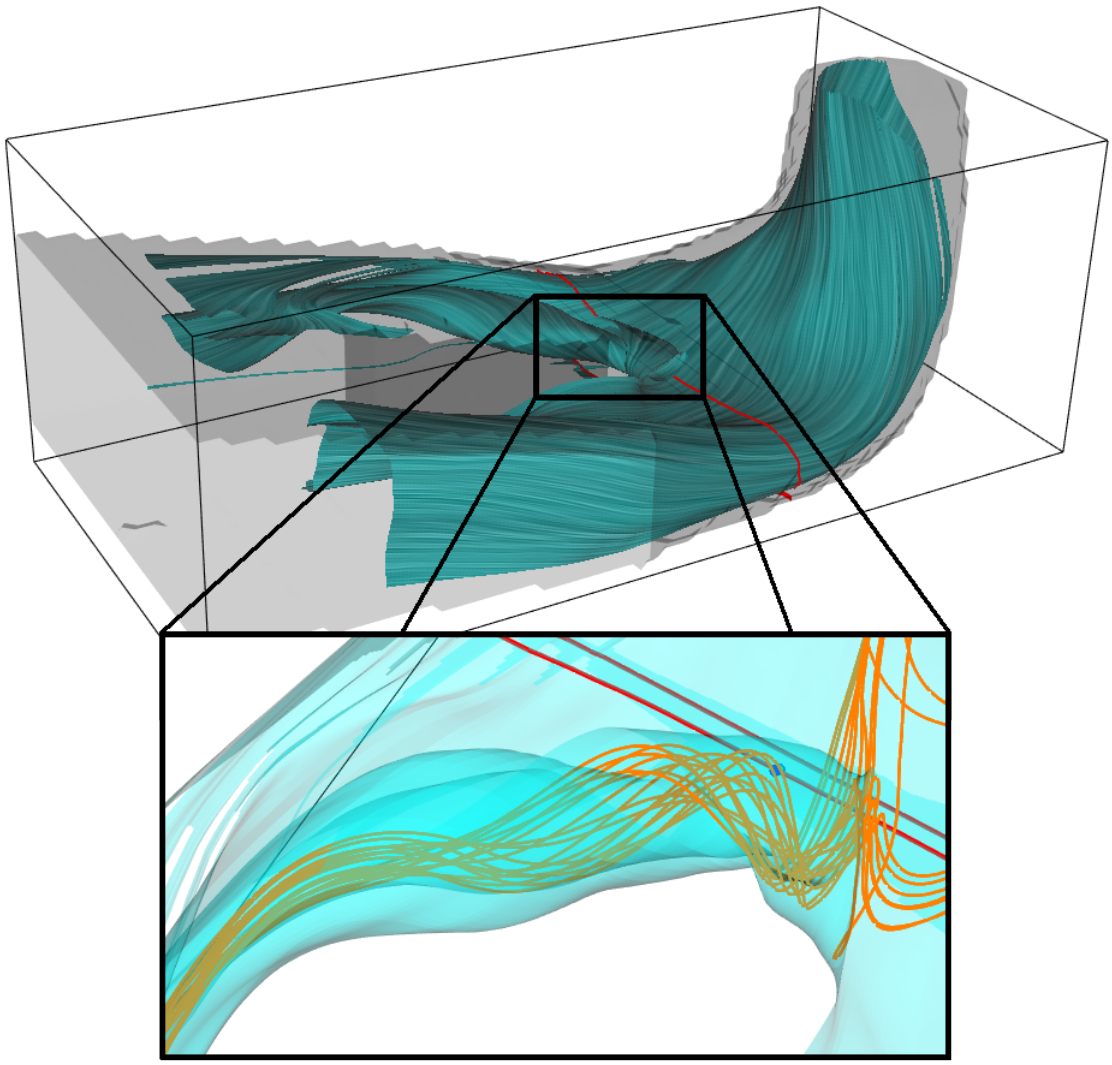


Figure 4.5: A single stream surface seeded from our seeding curve algorithm in the Francis Turbine flow field. The direction of the flow on the surface is indicated with LIC. Additionally, a magnified vortex is show with streamlines to give additional context.

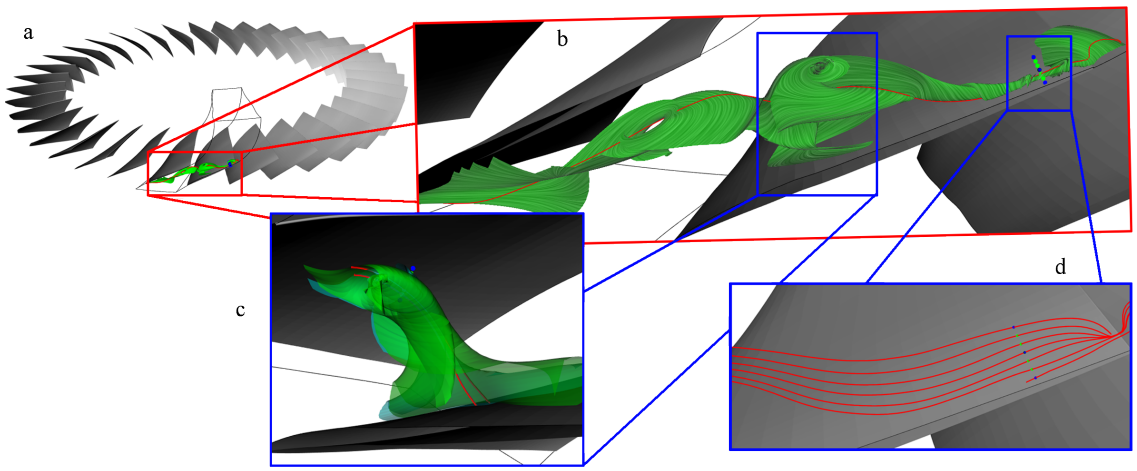


Figure 4.6: A stream surface visualization of the vortex breakdown behavior in the Jet Engine Compressor data set. (a) 36 blades of the jet engine compressor. (b) Stream surface painted with LIC that visualizes the vortex breakdown during stall conditions. (c) Magnified vortex shown in (b) visualized by an additional stream surface. (d) Seeding curves automatically generated by our algorithm near the region where streamlines are traditionally placed.

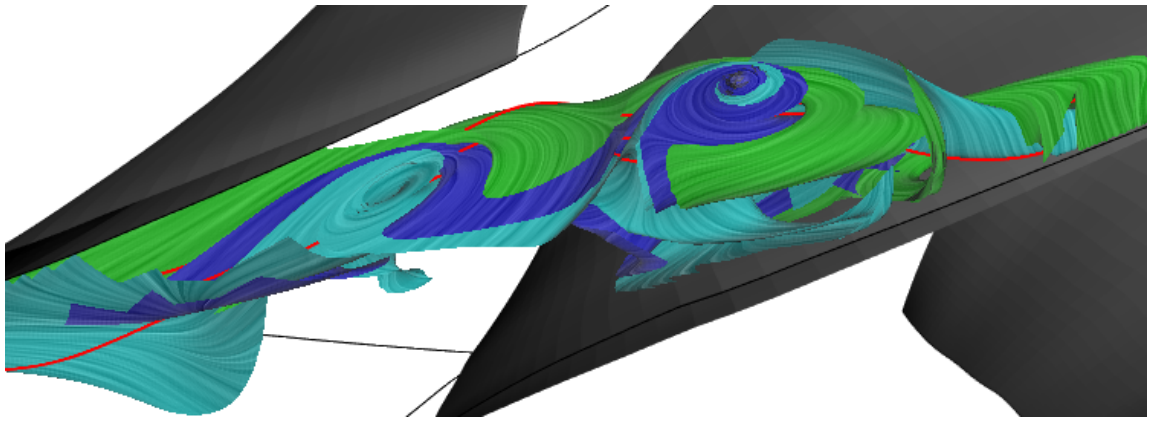
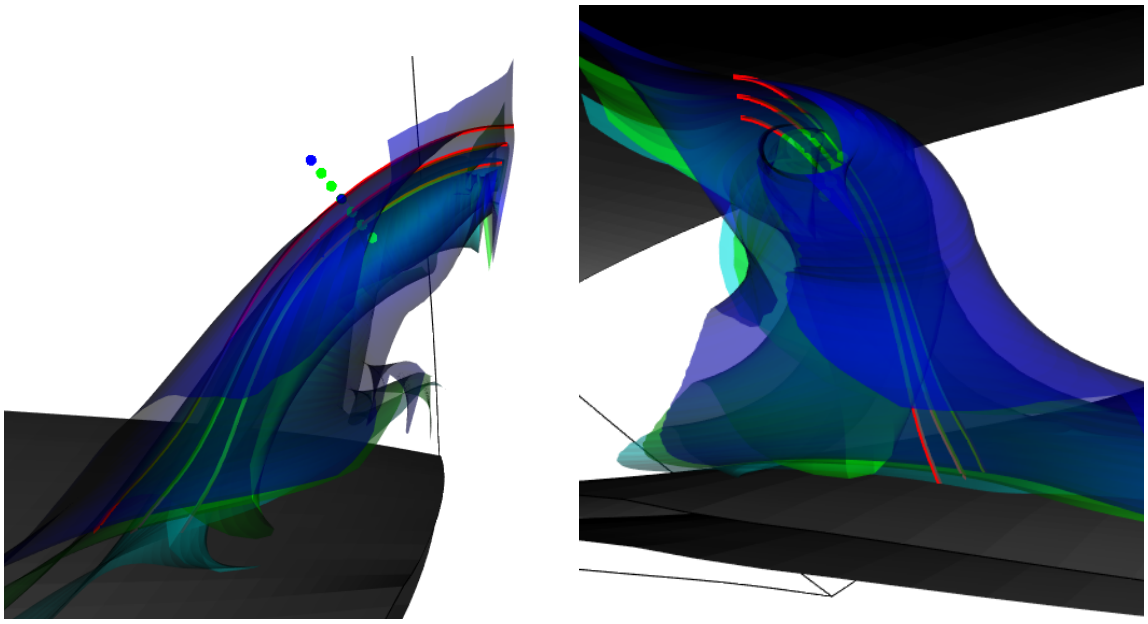


Figure 4.7: All of the stream surfaces in the Jet Engine Compressor data set generated from the seeding curves shown in Figure 4.6.



(a) (b)

Figure 4.8: Stream surfaces seeded downstream in the Jet Engine Compressor data set. (a) and (b) show that the vortex in this region attaches to the casing and blade. These stream surfaces visualize the same vortex shown in Figure 4.6.c.

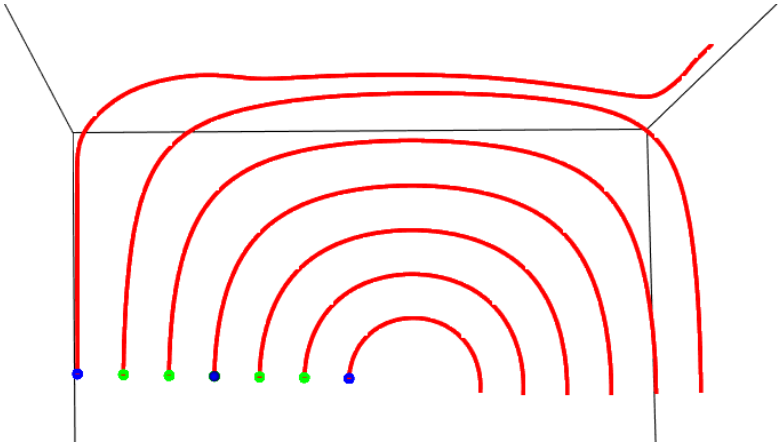


Figure 4.9: Seeding curves in the Solar Plume data set. The outermost seeding curve diverges from all the others and behaves unpredictably.

Bibliography

[1] Michael Bartoň, Jiří Kosinka, and Victor M. Calo. Stretch-minimising stream surfaces. *Graphical Models*, 79:12–22, may 2015.

[2] Andrea Brambilla and Helwig Hauser. Expressive seeding of multiple stream surfaces for interactive flow exploration. *Computers & Graphics*, 47:123–134, apr 2015.

[3] S. Camarri, M.-V. Salvetti, M. Buffoni, and A. Iollo. Simulation of the three-dimensional flow around a square cylinder between parallel walls at moderate Reynolds numbers. In *XVII Congresso di Meccanica Teorica ed Applicata*, 2005.

[4] A. Chaudhuri, Teng-Yok Lee, Han-Wei Shen, and R. Wenger. Exploring flow fields using space-filling analysis of streamlines. *Visualization and Computer Graphics, IEEE Transactions on*, 20(10):1392–1404, Oct 2014.

[5] Chun-Ming Chen, Soumya Dutta, Xiaotong Liu, Gregory Heinlein, Han-Wei Shen, and Jen-Ping Chen. Visualization and analysis of rotating stall for transonic jet engine simulation. *IEEE Transactions on Visualization and Computer Graphics*, 22(1):847–856, jan 2016.

[6] R.A. Crawfis and N. Max. Texture splats for 3D scalar and vector field visualization. In *Proceedings Visualization '93*. Institute of Electrical and Electronics Engineers (IEEE), 1993.

[7] M. Edmunds, R.S. Laramee, R. Malki, I. Masters, T.N. Croft, G. Chen, and E. Zhang. Automatic stream surface seeding: A feature centered approach. *Computer Graphics Forum*, 31(3pt2):1095–1104, jun 2012.

[8] Matt Edmunds, Robert S. Laramee, Guoning Chen, Nelson Max, Eugene Zhang, and Colin Ware. Surface-based flow visualization. *Computers & Graphics*, 36(8):974–990, dec 2012.

[9] Matt Edmunds, Robert S. Laramee, Guoning Chen, Eugene Zhang, and Nelson Max. Advanced, automatic stream surface seeding and filtering. *Theory and Practice of Computer Graphics*, 2012.

[10] Janick Martinez Esturo, Maik Schulze, Christian Rössl, and Holger Theisel. Global selection of stream surfaces. *Computer Graphics Forum*, 32(2pt1):113–122, may 2013.

[11] Kenneth Falconer. *Fractal Geometry: Mathematical Foundations and Applications*. Wiley, 2003.

[12] Christoph Garth, Xavier Tricoche, Tobias Salzbrunn, Tom Bobach, and Gerik Scheuermann. Surface techniques for vortex visualization, 2004.

[13] E. Heiberg, T. Ebbers, L. Wigstrom, and M. Karlsson. Three-dimensional flow characterization using vector pattern matching. *Visualization and Computer Graphics, IEEE Transactions on*, 9(3):313–319, July 2003.

[14] J.P.M. Hultquist. Constructing stream surfaces in steady 3D vector fields. In *Proceedings Visualization '92*. Institute of Electrical and Electronics Engineers (IEEE), 1992.

[15] M. Khoury and R. Wenger. On the fractal dimension of isosurfaces. *Visualization and Computer Graphics, IEEE Transactions on*, 16(6):1198–1205, Nov 2010.

[16] Peter Paul Klein. On the ellipsoid and plane intersection equation. *AM*, 03(11):1634–1640, 2012.

[17] Teng-Yok Lee, O. Mishchenko, Han-Wei Shen, and R. Crawfis. View point evaluation and streamline filtering for flow visualization. In *Visualization Symposium (PacificVis), 2011 IEEE Pacific*, pages 83–90, March 2011.

[18] K. Mahrous, J. Bennett, G. Scheuermann, B. Hamann, and K.I. Joy. Topological segmentation in three-dimensional vector fields. *Visualization and Computer Graphics, IEEE Transactions on*, 10(2):198–205, March 2004.

[19] Xiaoyang Mao, Y. Hatanaka, H. Higashida, and A. Imamiya. Image-guided streamline placement on curvilinear grid surfaces. In *Visualization '98. Proceedings*, pages 135–142, Oct 1998.

[20] S. Marchesin, Cheng-Kai Chen, C. Ho, and Kwan-Liu Ma. View-dependent streamlines for 3d vector fields. *Visualization and Computer Graphics, IEEE Transactions on*, 16(6):1578–1586, Nov 2010.

[21] T. McLoughlin, M.W. Jones, R.S. Laramee, R. Malki, I. Masters, and C.D. Hansen. Similarity measures for enhancing interactive streamline seeding. *IEEE Transactions on Visualization and Computer Graphics (TVCG)*, 19(8):1342–1353, 2013.

[22] A. Mebarki, P. Alliez, and O. Devillers. Farthest point seeding for efficient placement of streamlines. In *Visualization, 2005. VIS 05. IEEE*, pages 479–486, Oct 2005.

[23] Ronald Peikert and Filip Sadlo. Topologically relevant stream surfaces for flow visualization. In *Proceedings of the 2009 Spring Conference on Computer Graphics - SCCG '09*. Association for Computing Machinery (ACM), 2009.

[24] I Ari Sadarjoen and Frits H Post. Geometric methods for vortex extraction. In *Data Visualization99*, pages 53–62. Springer, 1999.

[25] I.A. Sadarjoen, F.H. Post, Bing Ma, D.C. Banks, and H.-G. Pagendarm. Selective visualization of vortices in hydrodynamic flows. In *Visualization '98. Proceedings*, pages 419–422, Oct 1998.

[26] G. Scheuermann, T. Bobach, H. Hagen, K. Mahrous, B. Hamann, K.I. Joy, and W. Kollmann. A tetrahedra-based stream surface algorithm. In *Proceedings Visualization, 2001. VIS '01.* Institute of Electrical & Electronics Engineers (IEEE), 2001.

[27] Dominic Schneider, Wieland Reich, Alexander Wiebel, and Gerik Scheuermann. Topology aware stream surfaces. *Computer Graphics Forum*, 29(3):1153–1161, aug 2010.

[28] M. Schulze, J. Martinez Esturo, T. Gnther, C. Rössl, H.-P. Seidel, T. Weinkauf, and H. Theisel. Sets of globally optimal stream surfaces for flow visualization. *Computer Graphics Forum*, 33(3):1–10, jun 2014.

[29] Greg Turk and David Banks. Image-guided streamline placement. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '96, pages 453–460, New York, NY, USA, 1996. ACM.

[30] J.J. van Wijk. Implicit stream surfaces. In *Proceedings Visualization '93*. Institute of Electrical and Electronics Engineers (IEEE), 1993.

[31] W. von Funck, T. Weinkauf, H. Theisel, and H.-P. Seidel. Smoke surfaces: An interactive flow visualization technique inspired by real-world flow experiments. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1396–1403, nov 2008.

[32] Wikipedia. Cantor set binary tree, 2017.

[33] Wikipedia. Koch curve, 2017.

[34] Lijie Xu, Teng-Yok Lee, and Han-Wei Shen. An information-theoretic framework for flow visualization. *Visualization and Computer Graphics, IEEE Transactions on*, 16(6):1216–1224, Nov 2010.

[35] Xiaohong Ye, D. Kao, and A. Pang. Strategy for seeding 3D streamlines. In *VIS 05. IEEE Visualization, 2005.* Institute of Electrical & Electronics Engineers (IEEE), 2005.

[36] Jialin Zhong, T.S. Huang, and R.J. Adrian. Extracting 3d vortices in turbulent fluid flow. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 20(2):193–199, Feb 1998.